Titre: Title:	Synthèse géométrique d'un manipulateur parallèle sphérique à trois degrés de liberté
Auteur: Author:	Stéphane Brunet
Date:	2003
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Brunet, S. (2003). Synthèse géométrique d'un manipulateur parallèle sphérique à trois degrés de liberté [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7234/

UNIVERSITÉ DE MONTRÉAL

SYNTHÈSE GÉOMÉTRIQUE D'UN MANIPULATEUR PARALLÈLE SPHÉRIQUE À TROIS DEGRÉS DE LIBERTÉ

STÉPHANE BRUNET
DÉPARTEMENT DE GÉNIE MÉCANIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE MÉCANIQUE)
DÉCEMBRE 2003

Your file Votre référence
ISBN: 0-612-89185-2
Our file Notre référence
ISBN: 0-612-89185-2

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

SYNTHÈSE GÉOMÉTRIQUE D'UN MANIPULATEUR PARALLÈLE SPHÉRIQUE À TROIS DEGRÉS DE LIBERTÉ

présenté par: BRUNET Stéphane

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. FORTIN Clément, Ph.D., président

M. BARON Luc, Ph.D., membre et directeur de recherche

M. CLOUTIER Guy, Doct., membre

RÉSUMÉ

Ce mémoire présente la synthèse géométrique de manipulateurs parallèles sphériques de topologie 3-RRR à trois degrés de liberté en rotation à l'aide d'algorithmes génétiques codés réels. L'application envisagée est une machine à commande numérique hybride à six axes composée de deux manipulateurs parallèles : un pour les trois degrés de liberté en rotation et un pour les trois degrés de liberté en translation. Le travail présenté porte uniquement sur la portion d'orientation. À partir du cahier des charges préliminaire dressé sont choisis les critères d'optimisation.

Le premier objectif est de décrire à l'aide d'un nombre minimal de paramètres l'ensemble des manipulateurs 3-RRR de géométrie sphérique. Douze paramètres indépendants ont été identifiés. À partir de ce paramétrage, la cinématique du manipulateur est étudiée et le modèle géométrique inverse est résolu.

Afin de mener une optimisation de l'ensemble des paramètres géométriques pour l'application envisagée, le calcul des espaces de travail des manipulateurs étudiés doit être traité. Cette tâche est réalisée de manière numérique en effectuant une discrétisation de l'espace à l'aide de 2^k-arbres qui sont une généralisation des arbres binaires, des *quadtrees* et des *octrees*. Une librairie générique a été développée en C++ pour traiter tous les cas de figure rencontrés.

L'usage exclusif du modèle géométrique inverse pour le calcul des espaces de travail ne permet pas de prendre en compte les collisions qui peuvent survenir entre les membrures du robot. Il est montré dans ce mémoire que la prise en compte des collisions est indispensable pour obtenir une précision acceptable dans le calcul des espaces de travail. Deux méthodes de détection de collision ont été intégrées au calcul d'espaces de travail. La première se base sur un modèle polyédrique tandis

que l'autre utilise un modèle filaire pour représenter les membrures du robot. Une librairie générique a été développée en C++ pour les modèles filaires ainsi qu'une méthode rapide de détection de collision. Les deux techniques sont aussi comparées.

L'objectif principal est de trouver les paramètres géométriques optimaux pour l'application envisagée. Cette tâche est réalisée à l'aide d'algorithmes génétiques codés réels qui sont tout à fait adaptés aux problèmes complexes ayant de nombreux paramètres et un domaine de recherche mal connu. Les critères d'optimisation utilisés portent uniquement sur des caractéristiques géométriques : taille et forme de l'espace de travail, dimensions du bâti, des jambes et de l'effecteur. Outre l'obtention d'un manipulateur optimal, le calcul a montré que les manipulateurs les plus performants sont symétriques.

En plus des librairies pour les 2^k-arbres et la détection de collision, deux types de logiciels ont été développés : un simulateur pour la cinématique du robot, le calcul et la visualisation des espaces de travail et des programmes d'optimisation utilisant les algorithmes génétiques.

ABSTRACT

This thesis presents the geometrical synthesis of spherical parallel manipulators of 3-RRR topology using real-coded genetic algorithms. The intended use of this kind of robot is an hybrid numerically-controlled machine tool with six degrees of freedom composed of two parallel manipulators : one for the orientation degrees of freedom and one for the translation degrees of freedom. The presented work consists exclusively in the orientation part. The initially established specifications of the machine tool are used in the choice of optimization criteria.

The first goal of this work is to describe the entire class of spherical 3-RRR manipulators using a minimal set of geometrical parameters. Twelve independent parameters have been identified. Using this parameterization, the kinematics is studied and the inverse kinematics problem is addressed.

In order to optimize the set of geometrical parameters for the intended use, workspaces computation of studied manipulators must be done. This task is handled numerically by discretizing space using 2^k-trees which are a generalization of bintrees, quadtrees and octrees. A generic C++ library has been developed to tackle all kinds of cases.

The possible collisions between manipulator legs are not taken into account when workspaces computation rely exclusively on inverse kinematics. It is shown in this work that collision detection must be implemented in order to obtain results of sufficient precision. Two techniques are presented and integrated to the workspace computation : one based on a polyhedral model and one using a wireframe model to represent manipulator links. A generic C++ library has been developed for wireframe representations as well as a fast collision query algorithm. The two presented

techniques are also compared.

The main goal of this work is to find the optimal geometrical parameters for the intended application. This is achieved by using real-coded genetic algorithms which are well-suited for complex problems with many parameters and a misunderstood search space. The optimizing criteria are exclusively based on geometrical characteristics : workspace size and shape, size of the base, the legs and the end effector. As a result, an optimal manipulator is found. Moreover, it is shown that the most efficient manipulators are symmetric.

In addition to the libraries for 2^k-trees and collision detection, two pieces of software have been developed : a simulator for the robot kinematics and the workspace computation and visualization and optimization programs using genetic algorithms.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ANNEXES

LISTE DES NOTATIONS ET DES SYMBOLES

$\mathcal{B}$	Désigne une base
$\mathbf{v}$	Désigne un vecteur
$\{\mathbf{w}\}_{\mathcal{B}}$	Précise dans quelle base il faut exprimer les coordonnées du vecteur
$\mathbf{A}$	Désigne une matrice de transformation
$\mathbf{R}_X(\alpha)$	Désigne la matrice de rotation d'angle α autour de l'axe X
$\mathbf{R}_Y(\alpha)$	Désigne la matrice de rotation d'angle α autour de l'axe Y
$\mathbf{R}_Z(\alpha)$	Désigne la matrice de rotation d'angle α autour de l'axe Z
$\mathrm{c}\,\alpha$	Désigne le cosinus de l'angle α
$\mathrm{s}\,\alpha$	Désigne le sinus de l'angle α

INTRODUCTION

Depuis quelques décennies, la robotique a pris une place considérable dans l'industrie manufacturière en général. Le secteur de l'automobile a toujours été l'exemple en matière de robotisation mais les industries d'électronique et de micro-électronique sont aussi de grandes utilisatrices de systèmes robotiques.

Discipline très variée, la robotique fait appel à des connaissances dans le domaine de la mécanique, de l'électrotechnique et de l'informatique et, bien que très développée dans l'industrie, fait encore l'objet d'une recherche très active, que ce soit dans le domaine du contrôle des robots ou bien de la modélisation cinématique et dynamique des manipulateurs.

On peut classer les manipulateurs dans trois catégories : les manipulateurs *sériels*, *parallèles* ou *hybrides*. Les manipulateurs sériels sont les plus connus et les plus utilisés dans l'industrie pour le moment. Ils sont formés d'une boucle cinématique ouverte (fig.0.1). Parmi les manipulateurs parallèles, le plus connu est très certainement la plateforme de GOUGH-STEWART (GOUGH et WHITEHALL, 1962; STEWART, 1965). Elle est couramment utilisée dans les simulateurs de vols tels que ceux fabriqués par la compagnie CAE. Les manipulateurs parallèles sont constitués de plusieurs chaînes cinématiques reliant la base à l'effecteur (fig.0.2). Ainsi, une ou plusieurs boucles cinématiques fermées sont créées. La famille des manipulateurs hybrides est la plus générale. En effet, la chaîne cinématique peut être constituée de plusieurs boucles fermées et/ou ouvertes. Un exemple est donné à la figure 0.3.

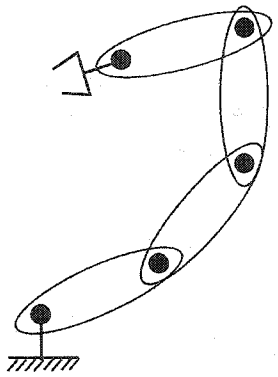


FIGURE 0.1 – Exemple de manipulateur sériel

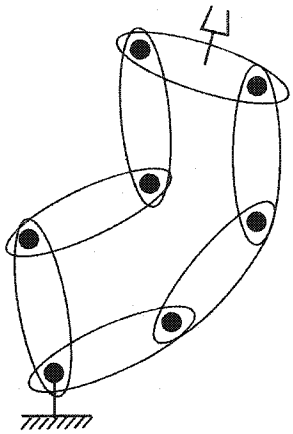


FIGURE 0.2 – Exemple de manipulateur parallèle

0.1 Généralités sur les manipulateurs parallèles

Les manipulateurs parallèles suscitent beaucoup d'intérêt depuis plusieurs années et sont au centre de nombreux projets de recherche universitaire ou industrielle. Les projets les plus avancés, menant à une application industrielle effectivement commercialisée, sont indéniablement les simulateurs de vol à nacelle mobile. Dans le domaine de la fabrication assistée par ordinateur, plusieurs fabricants de machines-outils à commande numérique se sont lancés dans la course à la réalisation de robots parallèles pour l'usinage.

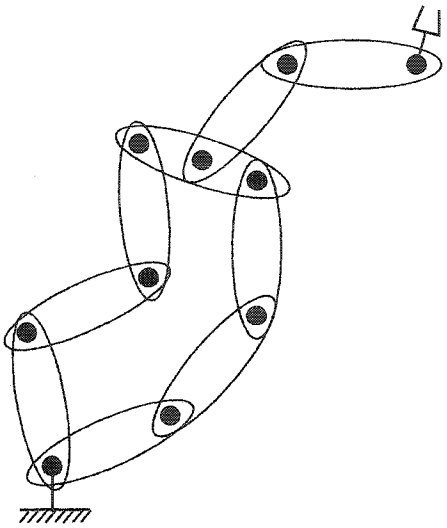


FIGURE 0.3 – Exemple de manipulateur hybride

Il est intéressant de comparer les manipulateurs sériels et parallèles afin d'expliquer l'engouement pour cette deuxième famille de manipulateurs. Le tableau 0.1 brosse un portrait général des caractéristiques des deux familles.

L'usage de robots parallèles dans les simulateurs de vol provient d'un rapport chargement maximum/masse très favorable. Remplacer la plateforme de GOUGH-STEWART à six degrés de liberté par un manipulateur sériel équivalent nécessiterait un bras incroyablement solide et des moteurs extrêmement puissants avec tous les problèmes que cela implique : grande inertie, beaucoup d'énergie consommée, masse et volume importants.

Plusieurs chaînes cinématiques en parallèle ont plusieurs avantages majeurs par rapport à une seule chaîne ouverte. D'abord, la rigidité du manipulateur s'en trouve accrue d'où une précision et une répétabilité meilleure. Dans un manipulateur sériel, les actionneurs sont disposés le long du bras. Ainsi, l'inertie du bras s'en trouve accrue. Il faut réduire la puissance des moteurs afin de limiter l'inertie, ce qui réduit la puissance du bras et la vitesse d'opération. Avec les manipulateurs parallèles,

TABLEAU 0.1 – Comparaison de quelques caractéristiques entre les manipulateurs sériels et parallèles (DANIALI, 1995)

Caractéristique	**Manipulateurs sériels**	**Manipulateurs parallèles**
Précision	inférieure	supérieure
Espace de travail	grand	petit
Rigidité	inférieure	supérieure
Rapport Chargement maxi / masse	inférieur	supérieur
Complexité de conception	faible	élevée
Inertie du système	supérieure	inférieure
Vitesse d'opération	inférieure	supérieure
Répétabilité	inférieure	supérieure
Densité des singularités	faible	élevée

il est souvent possible garder les actionneurs immobiles. Toutes les autres liaisons sont passives et l'inertie est alors réduite. Par ailleurs, la puissance des actionneurs peut être ajustée sans contrainte d'inertie donc les vitesses d'opération peuvent être plus élevées.

S'il n'existait que des avantages, les manipulateurs parallèles auraient déjà remplacé les robots sériels depuis longtemps. Malheureusement, les obstacles rencontrés dans la conception sont nombreux. En effet, l'existence de boucles cinématiques fermées complique beaucoup la cinématique et la dynamique déjà complexes pour les robots sériels. Par ailleurs, l'existence de plusieurs chaînes en parallèle augmente de beaucoup les risques de collision entre les membrures du manipulateur. L'espace de travail des manipulateurs parallèles est donc souvent beaucoup plus petit que celui d'un robot sériel. Enfin, les singularités déjà problématiques pour les robots sériels sont plus nombreuses pour les manipulateurs parallèles. Celles-ci sont à éviter à cause des pertes ou gains de degrés de liberté qui rendent le manipulateur difficile à commander voire même incontrôlable.

0.2 Choix d'un manipulateur en fonction de l'application envisagée

Dans le choix d'un manipulateur, il faut distinguer la *topologie* et la *géométrie* de celui-ci. En effet, la topologie est la description de la chaîne cinématique, c'est-à-dire le graphe représentant la manière dont sont interreliés les solides constituant le manipulateur. Il existe deux types de liaisons cinématiques dites *supérieures* si le contact entre deux solides est une droite ou un point et *inférieures* si le contact est surfacique. Les liaisons cinématiques inférieures sont au nombre de six : pivot (ou rotoïde), glissière (ou prismatique), hélicoïdale, appui-plan, sphérique et pivot glissant (ou cylindrique) respectivement représentées par les lettres R, P, H, E, S et C. Toutefois, parmi ces liaisons, seulement deux sont essentielles : la liaison rotoïde et la liaison prismatique. En effet, toutes les autres peuvent être obtenues à l'aide de combinaisons de ces deux liaisons (ANGELES, 1997).

La géométrie du manipulateur s'appuie évidemment sur la topologie de celui-ci mais elle ne fait que représenter comment sont disposées dans l'espace les différentes liaisons cinématiques. Il est ainsi possible d'obtenir des manipulateurs de même topologie mais aux géométries bien différentes (un exemple est donné dans la section 0.3).

Choisir un manipulateur parallèle en fonction de l'application envisagée n'est pas une chose facile car il existe une multitude de topologies et de géométries ayant plus ou moins de degrés de liberté. Traditionnellement, une topologie précise est sélectionnée puis une géométrie associée est alors optimisée en fonction des contraintes de l'application choisie. L'optimisation effectuée s'appelle alors de la synthèse géométrique (TREMBLAY, 1999; BERNIER, 2003; MAJOU, WENGER et CHABLAT, 2002).

Lorsque l'optimisation porte sur la topologie du manipulateur, il est alors ques-

tion de synthèse topologique, sujet de recherche encore moins exploré. Le premier problème rencontré est de trouver une manière de représenter la topologie d'un manipulateur. Ensuite, il faut que cette représentation soit intégrable dans un programme d'optimisation. La théorie des graphes a été une technique proposée dans la représentation de la topologie (DOBRJANSKYJ et FREUDENSTEIN, 1967). L'usage des groupes de LIE ont mené à la création du manipulateur de topologie Star (HERVE et SPARACINO, 1992; SPARACINO et HERVE, 1993). Très récemment, WANG, BARON et CLOUTIER (2003a, 2003b) ont proposé une approche prometteuse de représentation à la fois topologique et géométrique de manipulateurs à l'aide du concept de diagramme topographique.

0.3 Présentation du manipulateur sphérique à trois degrés de libertés

0.3.1 Caractéristiques topologiques et géométriques

La topologie des manipulateurs 3-RRR est présentée sur la figure 0.4. Les trois liaisons rotoïdes du bâti sont les seules qui ont besoin à être actionnées. L'avantage d'une telle approche est de pouvoir utiliser n'importe quel moteur sans modifier l'inertie des parties mobiles du manipulateur. Il est possible de démontrer que le nombre de degrés de libertés obtenus par une telle topologie est de trois (DANIALI, 1995).

L'usage d'une telle topologie en pratique mène à deux familles distinctes de manipulateurs : les manipulateur 3-RRR plans (Fig. 0.5) et sphériques (Fig. 0.6). Seulement une caractéristique géométrique les distingue : les axes des joints rotoïdes sont tous parallèles dans le cas des manipulateurs plans alors qu'ils sont tous concourants dans le cas des manipulateurs sphériques. La nature des degrés de libertés obtenus est donc différente : deux translations et une rotation pour les manipulateurs

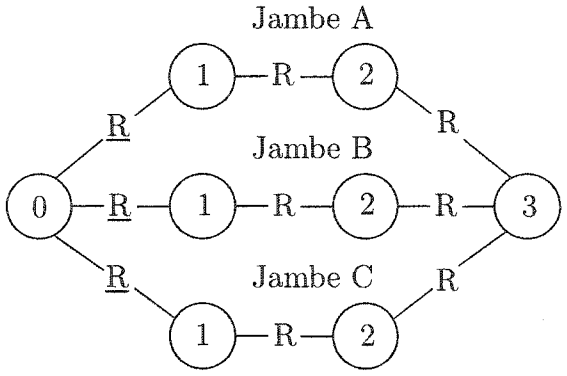


FIGURE 0.4 – Topologie des manipulateurs parallèles 3-RRR

plan, trois rotations pour les manipulateurs sphériques. La figure 0.7 présente les différents éléments constituants un manipulateurs sphérique 3-RRR tel qu'étudié dans ce mémoire. Les solides sont numérotés conformément à la figure 0.4.

0.3.2 Revue bibliographique et projets déjà réalisés

Le travail réalisé et publié sur les manipulateurs de topologie 3-RRR couvre les sujets suivants : la cinématique, la dynamique, l'optimisation et le contrôle ainsi que la réalisation de prototypes. La technique de résolution du modèle géométrique inverse semble avoir été trouvée par CRAVER (1989) puis reprise dans les travaux de GOSSELIN. Dans leur thèse de doctorat, DANIALI (1995) propose une représentation mathématique basée sur les *quaternions* tandis que FATTAH (1995) présente l'usage de liens flexibles pour les manipulateurs planaires.

La personne ayant le plus étudié les manipulateurs de topologie 3-RRR est indéniablement Clément GOSSELIN (1988) de l'Université Laval à Québec. Un travail étalé sur plus d'une décennie a mené à la réalisation de deux prototypes de robots sphériques 3-RRR : l'*œil agile* (fig. 8(a)), système d'orientation de caméra vidéo ultra-rapide et *SHaDe* (fig. 8(b)), poignée tridimensionnelle avec retour de force pour la téléopération. Ses nombreuses publications traitent des sujets suivants :

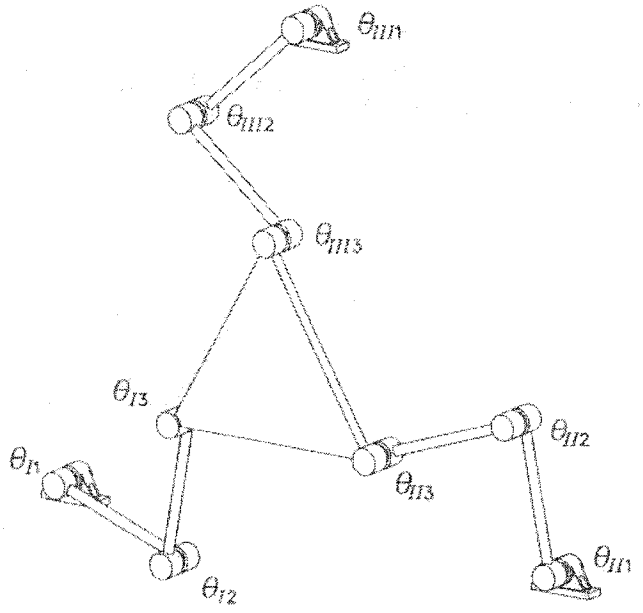


FIGURE 0.5 – Manipulateur parallèle plan (ANGELES, 1997)

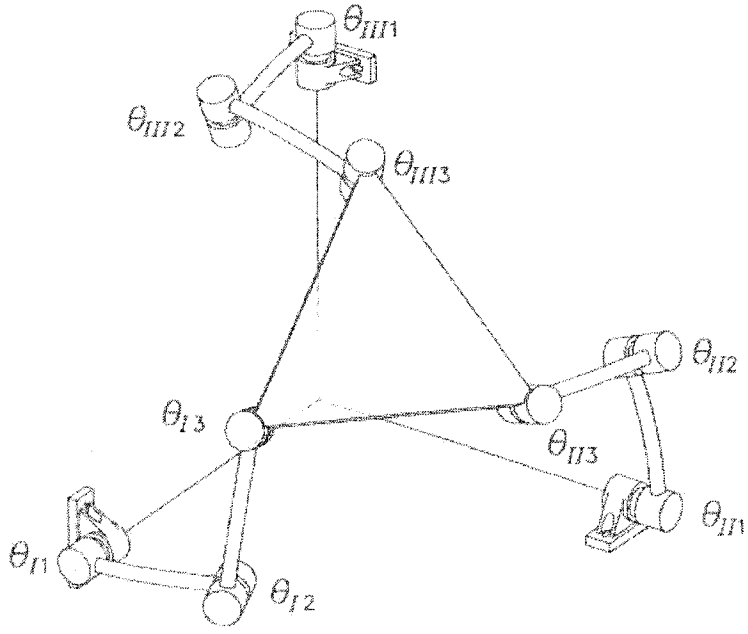


FIGURE 0.6 – Manipulateur parallèle sphérique (ANGELES, 1997)

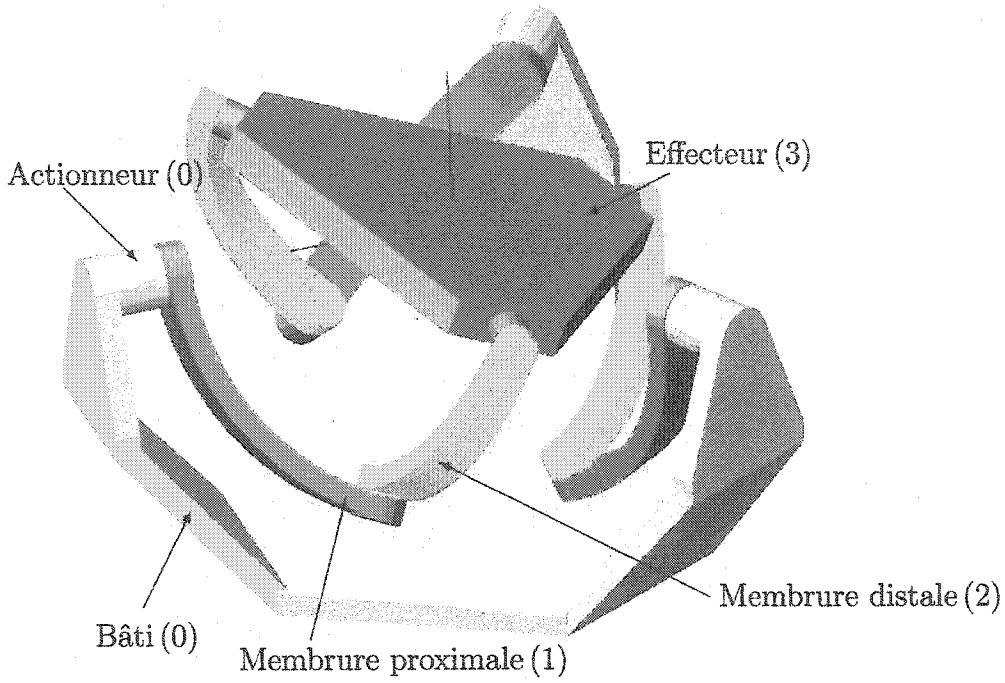


FIGURE 0.7 – Les éléments d'un manipulateur sphérique 3-RRR.

- le modèle géométrique inverse et le design optimal d'un manipulateur sphérique (GOSSELIN et ANGELES, 1989);
- le modèle géométrique direct des manipulateurs sphériques (GOSSELIN, SEFRIOUI et RICHARD, 1994a, 1994b);
- L'équilibrage statique (GOSSELIN, 1999);
- La conception de l'œil agile, sa cinématique, sa dynamique (GOSSELIN et LAVOIE, 1993; GOSSELIN, ST-PIERRE et GAGNÉ, 1996);
- L'analyse des obstructions dans l'œil agile (GOSSELIN et HAMEL, 1994);
- La conception de SHaDe (BIRGLEN et coll., 2002).

La synthèse de manipulateurs sphériques 3-RRR symétriques a déjà été étudiée par le passé (SABSABY, 2002).

Parmi les manipulateurs sphériques à trois degrés de liberté en rotation, d'autres topologies ont été étudiées. Dans sa thèse de doctorat, DANIALI (1995) propose un

manipulateur composé de deux triangles sphériques rigides reliés par des mécanismes PRP (prismatique actionné, rotoïde, prismatique). WIITALA et STANIŠIC (2000) présentent un prototype utilisant un manipulateur surcontraint (fig. 0.9).

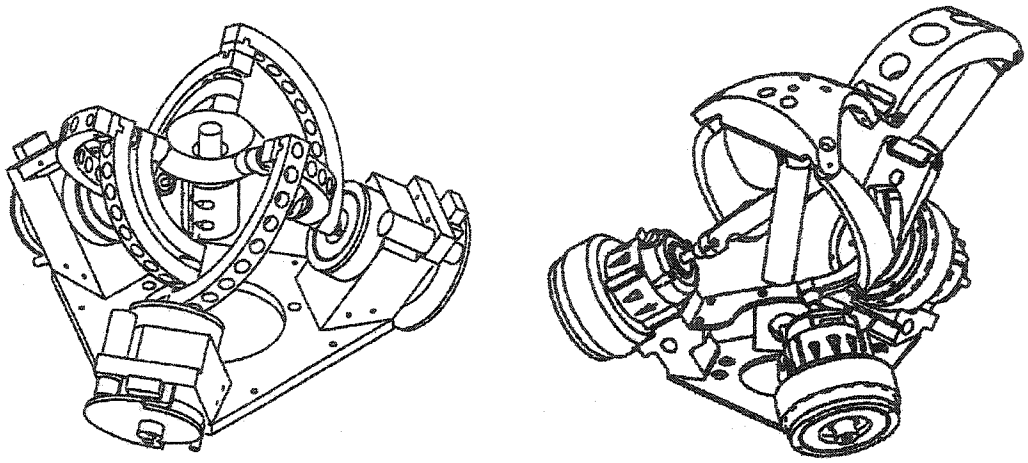
Toutefois, parmi toutes les publications consultées, seuls des manipulateurs 3-RRR sphériques symétriques sont étudiés. En fait, les optimisations réalisées portent sur un sous-ensemble de la famille des manipulateurs sphériques 3-RRR et non sur la famille au complet.

0.4 Objectifs du projet

Ce projet de recherche s'insert dans un projet plus ambitieux de machine-outil à commande numérique à six axes. Celle-ci sera en fait un robot hybride composé de deux manipulateurs parallèles à trois degrés de liberté, l'un pour l'orientation et l'autre pour la translation. La famille des manipulateurs sphériques 3-RRR a été choisie pour la partie orientation de la machine-outil. L'effecteur du robot servira de table d'usinage et orientera donc le brut par rapport à la broche. Ni la topologie et ni la géométrie du robot parallèle chargé de la translation n'ont été choisies jusqu'à présent. Cette partie de la machine permettra de positionner l'outil dans l'espace.

Le cahier des charges préliminaire a été défini comme suit :

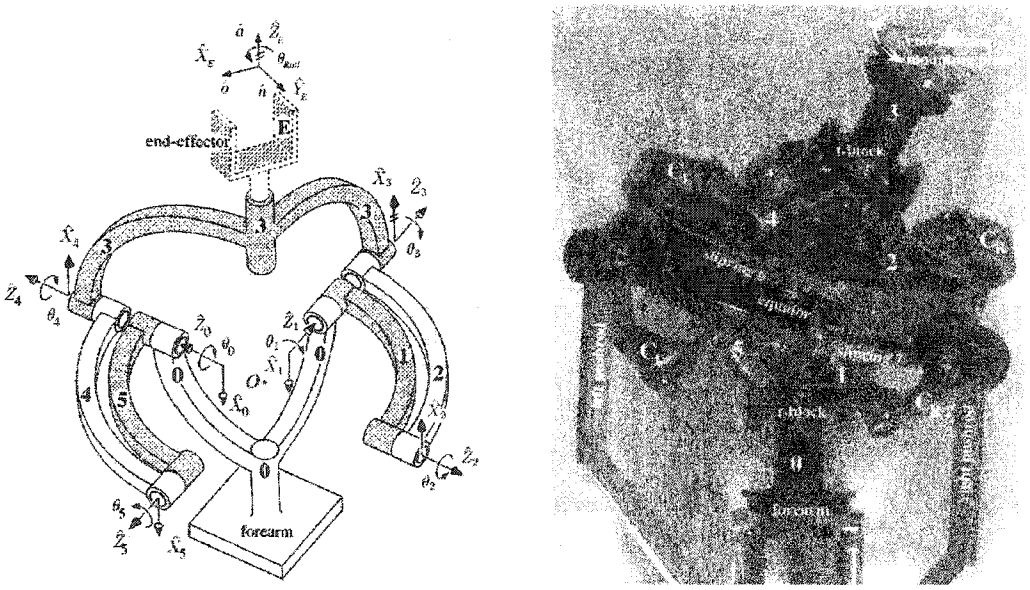
1. Amplitude des axes A et B : +/- 90°;
2. Amplitude de l'axe C suffisante pour éviter les obstructions;
3. Diamètre de la table : 400 mm avec rainures de fixation;
4. Centre de rotation localisé à 200 mm au-dessus de la table;
5. Vitesse de rotation maximale : 15 RPM;



(a) L'œil agile (GOSSELIN et HAMEL, 1994)

(b) SHaDe (BIRGLEN et coll., 2002)

FIGURE 0.8 – Deux réalisations concrètes : l'œil agile et SHaDe.



(a) schéma

(b) photo

FIGURE 0.9 – Le manipulateur de WIITALA et STANIŠIC (2000)

6. Accélération maximale : 900 RPM^2;
7. Capacité pièce : 25 kg.

La rotation de l'effecteur par rapport au centre de la sphère est aussi accompagnée d'une translation des points de l'effecteur. Placer le centre de rotation au-dessus de la table permet de limiter l'effet de translation, contrairement à un centre de rotation placé au-dessous de la table d'usinage. En effet, lorsqu'un brut est fixé sur la table, les points du solide ont tendance à se rapprocher du centre de rotation plutôt qu'à s'en éloigner.

CHAPITRE 1

PARAMÉTRAGE DES MANIPULATEURS SPHÉRIQUES

1.1 Avant-propos

Il n'existe pas une manière unique de choisir les paramètres décrivant totalement la géométrie des manipulateurs sphériques 3-RRR. Plusieurs méthodes ont été utilisées jusqu'à présent. La notation de DENAVIT-HARTENBERG (DENAVIT et HARTENBERG, 1955; ANGELES, 1997; CRAIG, 1989), qui a été créée pour les manipulateurs sériels, n'est pas pratique dans le cas d'un robot parallèle à cause de l'existence de boucles cinématiques fermées. Par ailleurs, elle est trop générale car le manipulateur étudié ne fait appel qu'à des rotations. Il faut donc trouver une formulation plus adaptée.

En vue d'effectuer une certaine optimisation des paramètres géométriques du manipulateur, une contrainte essentielle dans le choix du paramétrage est de trouver le nombre minimal de paramètres indépendants décrivant la famille des manipulateurs sphériques 3-RRR au complet. La formulation obtenue utilise douze paramètres indépendants. Des contraintes sont appliquées aux paramètres afin de garantir l'unicité du paramétrage, c'est-à-dire qu'un manipulateur donné ne peut être décrit qu'avec un seul ensemble de valeurs pour ces paramètres.

1.2 Notations utilisées

Afin de décrire clairement à quels éléments du manipulateur sont rattachés les paramètres présentés, une notation à base de chiffres et de lettres est utilisée. Les chiffres 0 à 3 sont associés respectivement au bâti, aux membrures proximales, distales et à l'effecteur. Afin de différentier les membrures proximales et distales des différentes jambes et leurs joints associés, les lettres A, B et C sont utilisées pour désigner chaque jambe.

Les vecteurs $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ désignent respectivement des vecteurs unitaires dans les directions X, Y et Z.

1.3 Positionnement des actionneurs – géométrie du bâti

Trois paramètres seulement sont nécessaires pour décrire la forme du bâti. Comme présentés à la figure 1.1, les angles non orientés α_{AB}, α_{AC} et α_{BC} sont utilisés pour définir la position relative des trois axes de rotation Z_0^A, Z_0^B et Z_0^C des actionneurs. Trois bases orthonormées $\mathcal{B}_0^k$, définies par les triplets de vecteurs unitaires $(\mathbf{u}_0^k,\mathbf{v}_0^k,\mathbf{w}_0^k)$, sont associées aux trois actionneurs. Les vecteurs unitaires $\mathbf{u}_0^k$ sont construits à partir des $\mathbf{w}_0^k$ de la manière suivante :

$$\mathbf{u}_0^A = \frac{\mathbf{w}_0^A \times \mathbf{w}_0^B}{\|\mathbf{w}_0^A \times \mathbf{w}_0^B\|} \tag{1.1}$$

$$\mathbf{u}_0^B = \frac{\mathbf{w}_0^B \times \mathbf{w}_0^C}{\|\mathbf{w}_0^B \times \mathbf{w}_0^C\|} \tag{1.2}$$

$$\mathbf{u}_0^C = \frac{\mathbf{w}_0^C \times \mathbf{w}_0^A}{\|\mathbf{w}_0^C \times \mathbf{w}_0^A\|} \tag{1.3}$$

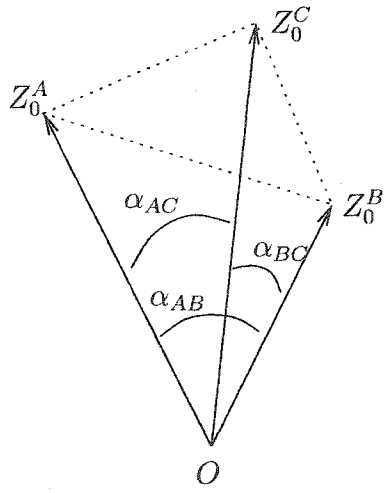


FIGURE 1.1 – Paramètres géométriques du bâti

Enfin, les vecteurs unitaires $\mathbf{v}_0^k$ sont construits de manière à obtenir des bases orthonormées directes, comme présenté sur la figure 1.2. Il faut noter que les expressions des vecteurs $\mathbf{u}_0^k$ sont valides à la seule condition que les vecteurs $\mathbf{w}_0^k$ ne soient pas colinéaires deux à deux.

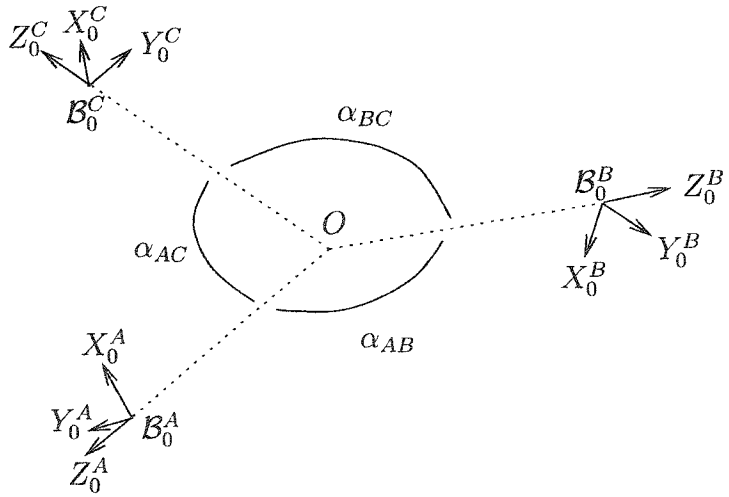


FIGURE 1.2 – Bases associées à chaque actionneur

Pour calculer les matrices de changement de base entre les trois actionneurs, il est nécessaire de calculer trois angles dièdres non orientés ζ_A, ζ_B et ζ_C. Ceux-ci sont

respectivement l'angle dièdre entre les plans $(O,\mathbf{w}_0^A,\mathbf{w}_0^B)$ et $(O,\mathbf{w}_0^A,\mathbf{w}_0^C)$, $(O,\mathbf{w}_0^A,\mathbf{w}_0^B)$ et $(O,\mathbf{w}_0^B,\mathbf{w}_0^C)$ et enfin $(O,\mathbf{w}_0^A,\mathbf{w}_0^C)$ et $(O,\mathbf{w}_0^B,\mathbf{w}_0^C)$.

Les valeurs des angles ζ_k s'obtiennent à l'aide des relations suivantes :

$$\zeta_A = \arccos\left(\frac{\cos\alpha_{BC} - \cos\alpha_{AB}\cos\alpha_{AC}}{\sin\alpha_{AB}\sin\alpha_{AC}}\right) \tag{1.4}$$

$$\zeta_B = \arccos\left(\frac{\cos\alpha_{AC} - \cos\alpha_{AB}\cos\alpha_{BC}}{\sin\alpha_{AB}\sin\alpha_{BC}}\right) \tag{1.5}$$

$$\zeta_C = \arccos\left(\frac{\cos\alpha_{AB} - \cos\alpha_{BC}\cos\alpha_{AC}}{\sin\alpha_{BC}\sin\alpha_{AC}}\right) \tag{1.6}$$

Ces expressions sont valides dans la mesure où les vecteurs $\mathbf{w}_0^k$ ne sont pas colinéaires deux à deux.

La matrice de transformation de la base $\mathcal{B}_0^A$ vers la base $\mathcal{B}_0^B$ est un produit de deux matrices de rotation, respectivement d'angle α_{AB} autour de l'axe X_0^A et d'angle $(\pi - \zeta_B)$ autour de l'axe Z_0^B :

$$\begin{aligned}\mathbf{D}_0^{AB} &= \mathbf{R}_{X_0^A}(\alpha_{AB})\,\mathbf{R}_{Z_0^B}(\pi-\zeta_B) \\ &= \begin{bmatrix} -\cos\zeta_B & -\sin\zeta_B & 0 \\ \sin\zeta_B\cos\alpha_{AB} & -\cos\zeta_B\cos\alpha_{AB} & -\sin\alpha_{AB} \\ \sin\zeta_B\sin\alpha_{AB} & -\cos\zeta_B\sin\alpha_{AB} & \cos\alpha_{AB} \end{bmatrix}\end{aligned} \tag{1.7}$$

De la même manière, la matrice de transformation de la base $\mathcal{B}_0^B$ vers la base $\mathcal{B}_0^C$ est le produit des deux matrices de rotation, respectivement d'angle α_{BC} autour

de l'axe X_0^B et d'angle $(\pi - \zeta_C)$ autour de l'axe Z_0^C :

$$\begin{aligned} \mathbf{D}_0^{BC} &= \mathbf{R}_{X_0^B}(\alpha_{BC})\ \mathbf{R}_{Z_0^C}(\pi - \zeta_C) \\ &= \begin{bmatrix} -\cos\zeta_C & -\sin\zeta_C & 0 \\ \sin\zeta_C\cos\alpha_{BC} & -\cos\zeta_C\cos\alpha_{BC} & -\sin\alpha_{BC} \\ \sin\zeta_C\sin\alpha_{BC} & -\cos\zeta_C\sin\alpha_{BC} & \cos\alpha_{BC} \end{bmatrix} \end{aligned} \tag{1.8}$$

Enfin, la matrice de transformation de la base $\mathcal{B}_0^A$ vers la base $\mathcal{B}_0^C$ est le produit des deux matrices de rotation, respectivement d'angle $(\zeta_A - \pi)$ autour de l'axe Z_0^A et d'angle $-\alpha_{AC}$ autour de l'axe X_0^C :

$$\begin{aligned} \mathbf{D}_0^{AC} &= \mathbf{R}_{Z_0^A}(\zeta_A - \pi)\ \mathbf{R}_{X_0^C}(-\alpha_{AC}) \\ &= \begin{bmatrix} -\cos\zeta_A & \sin\zeta_A\cos\alpha_{AC} & \sin\zeta_A\sin\alpha_{AC} \\ -\sin\zeta_A & -\cos\zeta_A\cos\alpha_{AC} & -\cos\zeta_A\sin\alpha_{AC} \\ 0 & -\sin\alpha_{AC} & \cos\alpha_{AC} \end{bmatrix} \end{aligned} \tag{1.9}$$

1.4 Positionnement des rotoïdes distaux – géométrie de l'effecteur

Un paramétrage identique à celui du bâti est mis en place. Les angles non orientés δ_{AB}, δ_{AC} et δ_{BC} définissent la position relative des rotoïdes distaux(Figure 1.3).

Les trois bases $\mathcal{B}_3^k$ orthonormées directes sont construites selon les même règles que celles utilisées pour le bâti, c'est à dire :

$$\mathbf{u}_3^A = \frac{\mathbf{w}_3^A \times \mathbf{w}_3^B}{\|\mathbf{w}_3^A \times \mathbf{w}_3^B\|} \tag{1.10}$$

$$\mathbf{u}_3^B = \frac{\mathbf{w}_3^B \times \mathbf{w}_3^C}{\|\mathbf{w}_3^B \times \mathbf{w}_3^C\|} \tag{1.11}$$

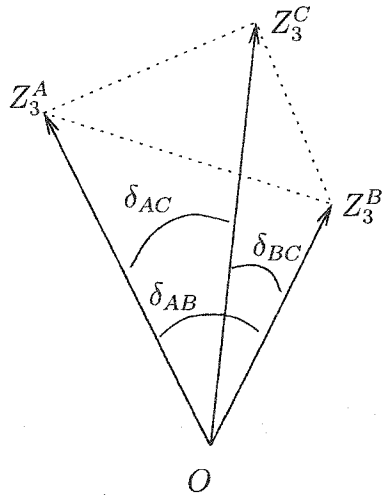


FIGURE 1.3 – Paramètres géométriques de l'effecteur

$$\mathbf{u}_3^C = \frac{\mathbf{w}_3^C \times \mathbf{w}_3^A}{\|\mathbf{w}_3^C \times \mathbf{w}_3^A\|} \tag{1.12}$$

Les vecteurs $\mathbf{v}_3^k$ sont ensuite construits pour compléter les bases orthonormées directes. Le résultat est présenté sur la figure 1.4. Il faut noter que les expressions des vecteurs $\mathbf{u}_3^k$ sont valides à la seule condition que les vecteurs $\mathbf{w}_3^k$ ne soient pas colinéaires deux à deux.

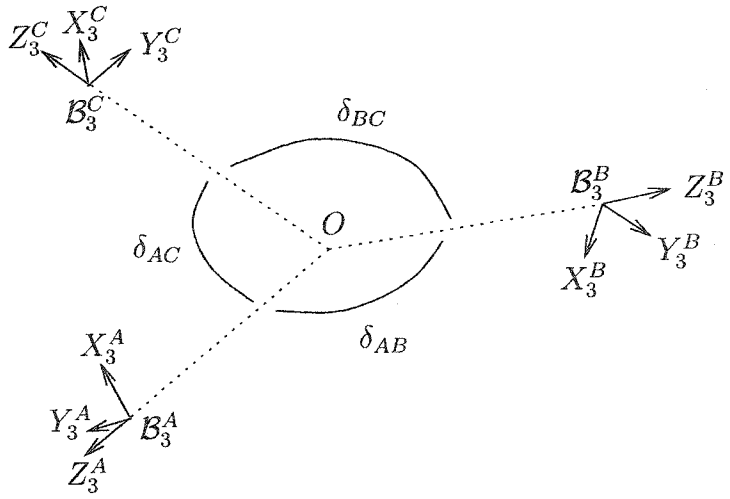


FIGURE 1.4 – Bases associées à chaque rotoïde distal

Trois angles non orientés ξ_k sont définis de la même manière que les angles ζ_k du bâti, c'est-à-dire comme les angles dièdres entre les plans $(O,\mathbf{w}_3^A,\mathbf{w}_3^B)$ et $(O,\mathbf{w}_3^A,\mathbf{w}_3^C)$, $(O,\mathbf{w}_3^A,\mathbf{w}_3^B)$ et $(O,\mathbf{w}_3^B,\mathbf{w}_3^C)$ et enfin $(O,\mathbf{w}_3^A,\mathbf{w}_3^C)$ et $(O,\mathbf{w}_3^B,\mathbf{w}_3^C)$.

Les expressions des cosinus des angles ξ_k correspondent exactement à celles écrites précédemment :

$$\xi_A = \arccos\left(\frac{\cos\delta_{BC} - \cos\delta_{AB}\cos\delta_{AC}}{\sin\delta_{AB}\sin\delta_{AC}}\right) \tag{1.13}$$

$$\xi_B = \arccos\left(\frac{\cos\delta_{AC} - \cos\delta_{AB}\cos\delta_{BC}}{\sin\delta_{AB}\sin\delta_{BC}}\right) \tag{1.14}$$

$$\xi_C = \arccos\left(\frac{\cos\delta_{AB} - \cos\delta_{BC}\cos\delta_{AC}}{\sin\delta_{BC}\sin\delta_{AC}}\right) \tag{1.15}$$

Ces expressions sont valides dans la mesure où les vecteurs $\mathbf{w}_3^k$ ne sont pas colinéaires deux à deux.

La matrice de transformation de $\mathcal{B}_3^A$ vers $\mathcal{B}_3^B$ est la composition de deux rotations, respectivement d'angle δ_{AB} autour de l'axe X_3^A et d'angle $(\pi - \xi_B)$ autour de l'axe Z_3^B :

$$\begin{aligned}\mathbf{D}_3^{AB} &= \mathbf{R}_{X_3^A}(\delta_{AB})\,\mathbf{R}_{Z_3^B}(\pi - \xi_B) \\ &= \begin{bmatrix} -\cos\xi_B & -\sin\xi_B & 0 \\ \sin\xi_B\cos\delta_{AB} & -\cos\xi_B\cos\delta_{AB} & -\sin\delta_{AB} \\ \sin\xi_B\sin\delta_{AB} & -\cos\xi_B\sin\delta_{AB} & \cos\delta_{AB} \end{bmatrix}\end{aligned} \tag{1.16}$$

De la même manière, la matrice de transformation de la base $\mathcal{B}_3^B$ vers la base $\mathcal{B}_3^C$ est le produit des deux matrices de rotation, respectivement d'angle α_{BC} autour

de l'axe X_3^B et d'angle $(\pi - \xi_C)$ autour de l'axe Z_0^C :

$$\begin{aligned} \mathbf{D}_3^{BC} &= \mathbf{R}_{X_3^B}(\delta_{BC})\, \mathbf{R}_{Z_3^C}(\pi - \xi_C) \\ &= \begin{bmatrix} -\cos\xi_C & -\sin\xi_C & 0 \\ \sin\xi_C \cos\delta_{BC} & -\cos\xi_C \cos\delta_{BC} & -\sin\delta_{BC} \\ \sin\xi_C \sin\delta_{BC} & -\cos\xi_C \sin\delta_{BC} & \cos\delta_{BC} \end{bmatrix} \end{aligned} \tag{1.17}$$

Enfin, la matrice de transformation de la base $\mathcal{B}_3^C$ vers la base $\mathcal{B}_3^A$ est le produit des deux matrices de rotation, respectivement d'angle $(\xi_A - \pi)$ autour de l'axe Z_3^A et d'angle $-\delta_{AC}$ autour de l'axe X_3^C :

$$\begin{aligned} \mathbf{D}_3^{AC} &= \mathbf{R}_{Z_3^A}(\xi_A - \pi)\, \mathbf{R}_{X_3^C}(-\delta_{AC}) \\ &= \begin{bmatrix} -\cos\xi_A & \sin\xi_A \cos\delta_{AC} & \sin\xi_A \sin\delta_{AC} \\ -\sin\xi_A & -\cos\xi_A \cos\delta_{AC} & -\cos\xi_A \sin\delta_{AC} \\ 0 & -\sin\delta_{AC} & \cos\delta_{AC} \end{bmatrix} \end{aligned} \tag{1.18}$$

1.5 Paramétrage des membrures

La figure 1.5 présente une jambe du manipulateur avec les paramètres géométriques correspondants. Les membrures proximales et distales sont numérotées respectivement 1 et 2.

Aux axes Z_1^k et Z_2^k, axes de rotation des joints proximal et intermédiaire de la jambe k, sont associés les vecteurs unitaires respectifs $\mathbf{w}_1^k$ et $\mathbf{w}_2^k$. Comme les axes Z_0^k et Z_1^k correspondent tous deux à l'axe de rotation du joint proximal, les vecteurs unitaires $\mathbf{w}_0^k$ et $\mathbf{w}_1^k$ sont donc identiques.

La base $\mathcal{B}_1^k$ associée à la membrure proximale de la jambe k est alors définie à l'aide

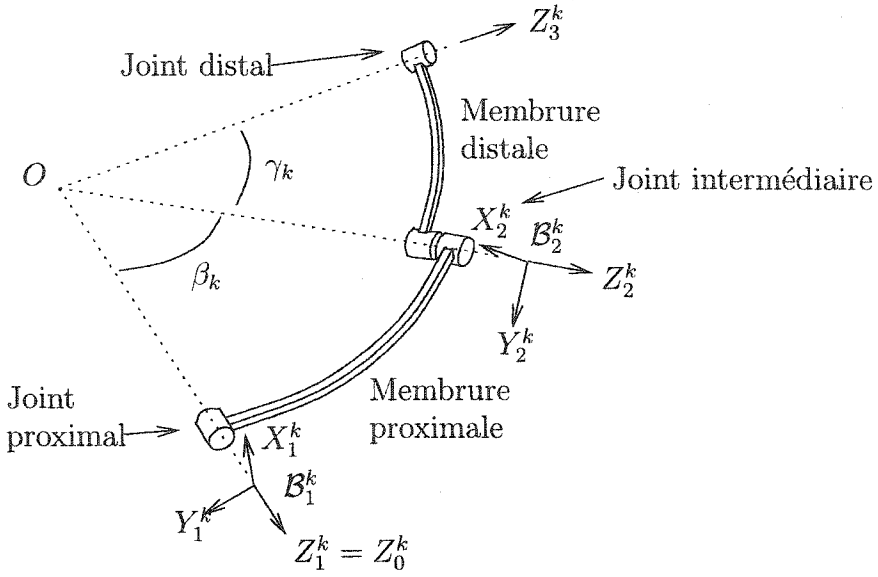


FIGURE 1.5 – Une jambe du manipulateur

du triplet de vecteurs unitaires $(\mathbf{u}_1^k, \mathbf{v}_1^k, \mathbf{w}_1^k)$ comme suit :

$$\mathbf{u}_1^k = \frac{\mathbf{w}_1^k \times \mathbf{w}_2^k}{\|\mathbf{w}_1^k \times \mathbf{w}_2^k\|} \tag{1.19}$$

et $\mathbf{v}_1^k$ construit de manière à ce que $\mathcal{B}_1^k$ soit orthonormale directe. De la même façon, la base $\mathcal{B}_2^k$ associée à la membrure distale de la jambe k est définie à l'aide du triplet de vecteurs unitaires $(\mathbf{u}_2^k, \mathbf{v}_2^k, \mathbf{w}_2^k)$ comme suit :

$$\mathbf{u}_2^k = \frac{\mathbf{w}_2^k \times \mathbf{w}_3^k}{\|\mathbf{w}_2^k \times \mathbf{w}_3^k\|} \tag{1.20}$$

et $\mathbf{v}_2^k$ construit de manière à ce que $\mathcal{B}_2^k$ soit orthonormale directe.

1.6 Contraintes associées aux paramètres de la base et de l'effecteur

Afin de garantir qu'un manipulateur quelconque ne puisse être décrit qu'avec un ensemble unique de valeurs pour les paramètres, il faut vérifier les contraintes

suivantes :

- Tous les angles α_{kl} et δ_{kl} doivent être inférieurs à π.
- La somme des angles α_{kl} doit être inférieure à 2π.
- La somme des angles δ_{kl} doit être inférieure à 2π.

1.7 Récapitulatif

Le bâti et l'effecteur sont décrits chacun par trois paramètres, respectivement, α_{AB}, α_{AB}, α_{AB} et δ_{AB}, δ_{AB}, δ_{AB}. Ceux-ci définissent l'angle entre les axes de rotation des joints proximaux, pour le bâti, et distaux, pour l'effecteur.

Chaque jambe est caractérisée par la dimension des membrures proximale et distale la constituant. La taille de la membrure proximale (respectivement distale) est définie grâce à l'angle β^k (respectivement γ^k) entre les axes de rotation des joints proximal et intermédiaire (respectivement intermédiaire et distal).

Douze paramètres sont donc nécessaires pour définir la géométrie la plus générale des manipulateurs 3-$\underline{\text{R}}$RR sphériques.

CHAPITRE 2

CINÉMATIQUE DES MANIPULATEURS SPHÉRIQUES 3-RRR

2.1 Introduction

La cinématique d'un manipulateur peut être résolue de deux manières différentes. Connaissant la posture (position et orientation) des différents liens, celle de l'effecteur est ensuite calculée : c'est le modèle géométrique direct. À l'inverse, lorsque la posture de l'effecteur est connue et que celle des différentes membrures est inconnue, il faut résoudre le modèle géométrique inverse.

Dans le cas des manipulateurs sériels, le modèle géométrique direct est trivial tandis que l'inverse pose plus de difficultés. Par contre, dans le cas des manipulateurs parallèles, l'existence de boucles cinématiques fermées complique grandement le problème. Le modèle géométrique direct est loin d'être facile à résoudre. Par contre, moyennant une technique de résolution progressive adaptée au manipulateur étudié, le modèle géométrique inverse se résout plus aisément.

La technique utilisée dans ce projet repose sur celles déjà publiée (CRAVER, 1989; GOSSELIN et ANGELES, 1989) mais adaptée à la formulation présentée précédemment. Afin d'éliminer une partie des inconnues, une expression basée sur la géométrie du manipulateur est utilisée. Le modèle géométrique inverse peut alors être résolu en deux temps : l'orientation des membrures proximales d'abord puis celle des membrures distales ensuite.

Le logiciel MATLAB a servi à générer toutes les équations symboliques nécessaires à la résolution de la cinématique à l'aide du paramétrage proposé dans ce mémoire.

Le fichier source est disponible à l'annexe II.

2.2 Inconnues cinématiques

Toutes les liaisons cinématiques entre les différents corps rigides constituant le manipulateur sont des rotoïdes. La position de la membrure proximale de la jambe k par rapport au bâti est définie à l'aide de l'angle θ_1^k (joint actionné de la jambe k). De manière pratique, l'angle θ_1^k correspond à un angle de rotation entre la base $\mathcal{B}_0^k$ et la base $\mathcal{B}_1^k$ par rapport à l'axe des Z. De la même façon, pour chaque jambe k du manipulateur, la position de la membrure distale est définie par rapport à la membrure proximale à l'aide de l'angle θ_2^k. Celui-ci correspond en pratique à l'angle de rotation selon l'axe des Z de la base $\mathcal{B}_2^k$ par rapport à la base $\mathcal{B}_1^k$.

Il serait possible de définir trois autres angles θ_3^k entre chaque membrure distale et l'effecteur. En effet, les angles θ_3^k pourraient être calculés à partir des valeurs obtenues pour les angles θ_1^k et θ_2^k mais ne servent à rien en pratique.

À l'aide de ces angles, il est possible de définir les matrices de transformation $\mathbf{Q}_{ij}^k$ de la base $\mathcal{B}_i^k$ vers la base $\mathcal{B}_j^k$:

$$\mathbf{Q}_{01}^k \equiv \mathbf{R}_Z(\theta_1^k) \tag{2.1a}$$

$$\mathbf{Q}_{12}^k \equiv \mathbf{R}_X(\beta_k)\,\mathbf{R}_Z(\theta_2^k) \tag{2.1b}$$

$$\mathbf{Q}_{23}^k \equiv \mathbf{R}_X(\gamma_k)\,\mathbf{R}_Z(\theta_3^k) \tag{2.1c}$$

2.3 Formulation du modèle géométrique inverse

Dans le cas du manipulateur sphérique étudié, la posture consiste seulement en une orientation de l'effecteur dans l'espace. Le problème géométrique inverse s'énonce alors de la manière suivante : connaissant l'orientation de l'effecteur, il faut calculer les valeurs des angles θ_1^k et θ_2^k.

La formulation du modèle géométrique inverse se base sur les trois angles d'Euler Z-X-Z pour définir l'orientation de l'effecteur. Bien que l'utilisation des angles d'Euler ne soit pas exempte de singularités mathématiques (lorsque la valeur du deuxième angle est nulle), l'influence de ces singularités est négligeable dans le cadre du calcul des espaces de travail de ce manipulateur, pour les raisons présentées dans le chapitre 3.

Les principales étapes dans la résolution du modèle géométrique inverse sont brièvement présentées ci-dessous et expliquées plus en détails dans les prochains paragraphes :

1. connaissant l'orientation de l'effecteur, calculer les coordonnées des vecteurs $\mathbf{w}_3^k$ dans une base de référence;
2. exprimer les coordonnées des vecteurs $\mathbf{w}_2^k$ dans cette même base à l'aide des inconnues θ_1^k;
3. utiliser l'expression suivante :

$$\mathbf{w}_2^k \cdot \mathbf{w}_3^k = \cos \gamma_k \tag{2.2}$$

   ce qui a pour effet de générer un système de trois équations à trois inconnues θ_1^k, non linéaires, en sinus et cosinus des angles θ_1^k;
4. exprimer les coordonnées des vecteurs $\mathbf{w}_3^k$ à l'aide des inconnues θ_2^k et des

valeurs obtenues pour les angles θ_1^k afin d'obtenir la valeurs des trois dernières inconnues restantes θ_2^k.

2.3.1 Choix d'une base de référence

Le premier choix comme base de référence était la base $\mathcal{B}_0^A$. L'orientation de l'effecteur était définie par rapport à cette base en orientant $\mathcal{B}_3^A$. Bien que ce choix mène à une formulation relativement simple du modèle géométrique inverse, il a été révisé par la suite afin de faciliter l'interprétation de la forme de l'espace de travail et la construction de critères d'optimisation plus simples.

La base $\mathcal{B}_0$ est définie par le triplet de vecteurs unitaires $(\mathbf{u}_0,\mathbf{v}_0,\mathbf{w}_0)$. Comme la géométrie du manipulateur peut varier, la base $\mathcal{B}_0$ doit être déduite des paramètres géométriques du manipulateur. Il a été décidé de l'orienter de manière à ce que son axe Z soit à une position intermédiaire par rapport aux axes de rotation des joint proximaux du bâti; c'est-à-dire que le vecteur unitaire $\mathbf{w}_0$ est défini comme le vecteur-somme normé des vecteurs $\mathbf{w}_0^k$ représentant la direction de l'axe de rotation des joints proximaux :

$$\mathbf{w}_0 = \frac{\mathbf{w}_0^A + \mathbf{w}_0^B + \mathbf{w}_0^C}{\|\mathbf{w}_0^A + \mathbf{w}_0^B + \mathbf{w}_0^C\|} \tag{2.3}$$

Si les trois vecteurs $\mathbf{w}_0^k$ sont coplanaires, le vecteur $\mathbf{w}_0$ se construit alors comme suit :

$$\mathbf{w}_0 = \frac{\mathbf{w}_0^A \times \mathbf{w}_0^B}{\|\mathbf{w}_0^A \times \mathbf{w}_0^B\|} \tag{2.4}$$

Le vecteur $\mathbf{v}_0$ est défini par la relation suivante :

$$\mathbf{v}_0 = \frac{\mathbf{w}_0 \times \mathbf{w}_0^A}{\|\mathbf{w}_0 \times \mathbf{w}_0^A\|} \tag{2.5}$$

Enfin, le vecteur $\mathbf{u}_0$ est construit de manière à obtenir une base orthonormale directe. En pratique, cette définition de $\mathcal{B}_0$, avec celle présentée pour $\mathcal{B}_3$ à la section 2.3.2, va permettre d'obtenir une position de référence plus naturelle de l'effecteur par rapport au bâti. L'analyse des résultats de calculs d'espaces de travail sera alors plus simple.

Il est possible de définir la matrice de transformation $\mathbf{D}_0$ de la base $\mathcal{B}_0$ vers la base $\mathcal{B}_0^A$:

$$\begin{aligned} \mathbf{D}_0 &= \mathbf{R}_X(\alpha_0^A)\,\mathbf{R}_Z(\pi-\zeta_0) \\ &= \begin{bmatrix} -\cos\zeta_0 & -\sin\zeta_0 & 0 \\ \sin\zeta_0\cos\alpha_0^A & -\cos\zeta_0\cos\alpha_0^A & -\sin\alpha_0^A \\ \sin\zeta_0\sin\alpha_0^A & -\cos\zeta_0\sin\alpha_0^A & \cos\alpha_0^A \end{bmatrix} \end{aligned} \tag{2.6}$$

avec

$$\alpha_0^A = \arccos\left(\frac{1+\cos\alpha_{AB}+\cos\alpha_{AC}}{\sqrt{3+2\times(\cos\alpha_{AB}+\cos\alpha_{AC}+\cos\alpha_{BC})}}\right) \tag{2.7}$$

et

$$\zeta_0 = \arccos\left(\frac{\cos\alpha_0^B-\cos\alpha_{AB}\cos\alpha_0^A}{\sin\alpha_{AB}\sin\alpha_0^A}\right) \tag{2.8}$$

et

$$\alpha_0^B = \arccos\left(\frac{1+\cos\alpha_{AB}+\cos\alpha_{BC}}{\sqrt{3+2\times(\cos\alpha_{AB}+\cos\alpha_{AC}+\cos\alpha_{BC})}}\right) \tag{2.9}$$

2.3.2 Calcul des coordonnées des vecteurs $\mathbf{w}_3^k$ à l'aide des angles d'Euler

Une base $\mathcal{B}_3$ rattachée à l'effecteur permet de définir l'orientation de l'effecteur par rapport à la base $\mathcal{B}_0$. Elle est construite de la même manière que la base $\mathcal{B}_0$ à l'aide d'un triplet de vecteurs unitaires ($\mathbf{u}_3$,$\mathbf{v}_3$,$\mathbf{w}_3$). Le vecteur $\mathbf{w}_3$ est défini de la manière

suivante :

$$\mathbf{w}_3 = \frac{\mathbf{w}_3^A + \mathbf{w}_3^B + \mathbf{w}_3^C}{\|\mathbf{w}_3^A + \mathbf{w}_3^B + \mathbf{w}_3^C\|} \tag{2.10}$$

Si les trois vecteurs $\mathbf{w}_3^k$ sont coplanaires, le vecteur $\mathbf{w}_3$ se construit alors comme suit :

$$\mathbf{w}_3 = \frac{\mathbf{w}_3^A \times \mathbf{w}_3^B}{\|\mathbf{w}_3^A \times \mathbf{w}_3^B\|} \tag{2.11}$$

Le vecteur $\mathbf{v}_3$ est défini par la relation suivante :

$$\mathbf{v}_3 = \frac{\mathbf{w}_3 \times \mathbf{w}_3^A}{\|\mathbf{w}_3 \times \mathbf{w}_3^A\|} \tag{2.12}$$

Enfin, le vecteur $\mathbf{u}_3$ est construit de manière à obtenir une base orthonormale directe. La définition de la base $\mathcal{B}_3$ permet de compléter celle de la base $\mathcal{B}_0$ vue dans la section 2.3.1.

Il est ainsi possible de définir la matrice de transformation $\mathbf{D}_3$ de la base $\mathcal{B}_3$ vers la base $\mathcal{B}_3^A$:

$$\begin{aligned} \mathbf{D}_3 &= \mathbf{R}_X(\delta_3^A)\, \mathbf{R}_Z(\pi - \xi_3) \\ &= \begin{bmatrix} -\cos\xi_3 & -\sin\xi_3 & 0 \\ \sin\xi_3 \cos\delta_3^A & -\cos\xi_3 \cos\delta_3^A & -\sin\delta_3^A \\ \sin\xi_3 \sin\delta_3^A & -\cos\xi_3 \sin\delta_3^A & \cos\delta_3^A \end{bmatrix} \end{aligned} \tag{2.13}$$

avec

$$\delta_3^A = \arccos\left(\frac{1 + \cos\delta_{AB} + \cos\delta_{AC}}{\sqrt{3 + 2 \times (\cos\delta_{AB} + \cos\delta_{AC} + \cos\delta_{BC})}}\right) \tag{2.14}$$

et

$$\xi_3 = \arccos\left(\frac{\cos\delta_3^B - \cos\delta_{AB}\cos\delta_3^A}{\sin\delta_{AB}\sin\delta_3^A}\right) \tag{2.15}$$

et

$$\delta_3^B = \arccos\left(\frac{1 + \cos\delta_{AB} + \cos\delta_{BC}}{\sqrt{3 + 2 \times (\cos\delta_{AB} + \cos\delta_{AC} + \cos\delta_{BC})}}\right) \tag{2.16}$$

L'orientation de l'effecteur par rapport à la base $\mathcal{B}_0$ est donnée à l'aide des angles d'Euler Z-X-Z, notés respectivement ψ_1, ψ_2 et ψ_3. La matrice de transformation $\mathbf{Q}_{03}$ de la base $\mathcal{B}_0$ vers la base $\mathcal{B}_3$ s'écrit alors :

$$\begin{aligned}\mathbf{Q}_{03} &\equiv \mathbf{R}_Z(\psi_1)\,\mathbf{R}_X(\psi_2)\,\mathbf{R}_Z(\psi_3) \\ &= \begin{bmatrix} \mathrm{c}\,\psi_1\,\mathrm{c}\,\psi_3 - \mathrm{s}\,\psi_1\,\mathrm{c}\,\psi_2\,\mathrm{s}\,\psi_3 & -\mathrm{c}\,\psi_1\,\mathrm{s}\,\psi_3 - \mathrm{s}\,\psi_1\,\mathrm{c}\,\psi_3\,\mathrm{c}\,\psi_2 & \mathrm{s}\,\psi_1\,\mathrm{s}\,\psi_2 \\ \mathrm{s}\,\psi_1\,\mathrm{c}\,\psi_3 + \mathrm{c}\,\psi_1\,\mathrm{c}\,\psi_2\,\mathrm{s}\,\psi_3 & -\mathrm{s}\,\psi_1\,\mathrm{s}\,\psi_3 + \mathrm{c}\,\psi_1\,\mathrm{c}\,\psi_2\,\mathrm{c}\,\psi_3 & -\mathrm{c}\,\psi_1\,\mathrm{s}\,\psi_2 \\ \mathrm{s}\,\psi_3\,\mathrm{s}\,\psi_2 & \mathrm{s}\,\psi_2\,\mathrm{c}\,\psi_3 & \mathrm{c}\,\psi_2 \end{bmatrix}\end{aligned} \tag{2.17}$$

L'expression des coordonnées des vecteurs $\mathbf{w}_3^k$ dans la base $\mathcal{B}_0$ s'obtient grâce à l'expression suivante :

$$\begin{aligned}\{\mathbf{w}_3^k\}_{\mathcal{B}_0} &= \mathbf{Q}_{03}\,\mathbf{D}_0^k\,\{\mathbf{w}_3^k\}_{\mathcal{B}_3^k} \\ &= \mathbf{Q}_{03}\,\mathbf{D}_0^k\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T\end{aligned} \tag{2.18}$$

avec

$$\mathbf{D}_3^A \equiv \mathbf{D}_3 \tag{2.19a}$$

$$\mathbf{D}_3^B \equiv \mathbf{D}_3\,\mathbf{D}_3^{AB} \tag{2.19b}$$

$$\mathbf{D}_3^C \equiv \mathbf{D}_3\,\mathbf{D}_3^{AC} \tag{2.19c}$$

2.3.3 Expression des coordonnées des vecteurs $\mathbf{w}_2^k$ en fonction des angles θ_1^k

L'expression des coordonnées des vecteurs $\mathbf{w}_2^k$ en fonction des angles θ_1^k s'obtient à l'aide de l'expression suivante :

$$\begin{aligned} \{\mathbf{w}_2^k\}_{\mathcal{B}_0} &= \mathbf{D}_0^k \, \mathbf{Q}_{01}^k \, \mathbf{Q}_{12}^k \, \{\mathbf{w}_2^k\}_{\mathcal{B}_2^k} \\ &= \mathbf{D}_0^k \, \mathbf{Q}_{01}^k \, \mathbf{Q}_{12}^k \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \end{aligned} \tag{2.20}$$

avec

$$\mathbf{D}_0^A \equiv \mathbf{D}_0 \tag{2.21a}$$

$$\mathbf{D}_0^B \equiv \mathbf{D}_0 \, \mathbf{D}_0^{AB} \tag{2.21b}$$

$$\mathbf{D}_0^C \equiv \mathbf{D}_0 \, \mathbf{D}_0^{AC} \tag{2.21c}$$

2.3.4 Élimination et résolution du système d'équations non linéaires en sinus et en cosinus

Pour chaque jambe k, l'expression suivante

$$\{\mathbf{w}_2^k\}_{\mathcal{B}_0} \cdot \{\mathbf{w}_3^k\}_{\mathcal{B}_0} = \cos\gamma_k \tag{2.22}$$

mène à une expression en sinus et en cosinus de la forme suivante :

$$A^k \cos\theta_1^k + B^k \sin\theta_1^k + C^k = 0 \tag{2.23}$$

qui peut être transformée en une équation linéaire du second degré en $\tan\frac{\theta_1^k}{2}$ (voir l'annexe I) :

$$\frac{C^k - A^k}{2}\left(\tan\frac{\theta_1^k}{2}\right)^2 + B^k \tan\frac{\theta_1^k}{2} + \frac{A^k + C^k}{2} = 0 \tag{2.24}$$

La résolution des trois équations indépendantes, obtenues à partir de l'expression ci-dessus pour $k = A$, B et C, mène à trois cas possibles, pour chaque jambe :

- Aucune solution : la géométrie du manipulateur ne permet pas d'atteindre cette orientation. De manière pratique, ceci correspond à une jambe trop courte.
- Deux solutions : la jambe peut avoir deux positions différentes pour atteindre la même orientation.
- Une seule solution : la jambe est en pleine extension.

2.3.5 Expression des coordonnées des vecteurs $\mathbf{w}_3^k$ et calcul des inconnues restantes θ_2^k

À l'aide des valeurs obtenues pour les angles θ_1^k et des coordonnées des vecteurs $\mathbf{w}_3^k$ calculées à l'aide de l'équation 2.18, l'expression suivante des vecteurs $\mathbf{w}_3^k$:

$$\begin{aligned} \{\mathbf{w}_3^k\}_{\mathcal{B}_0} &= \mathbf{D}_0^k\, \mathbf{Q}_{01}^k\, \mathbf{Q}_{12}^k\, \mathbf{Q}_{23}^k\, \{\mathbf{w}_3^k\}_{\mathcal{B}_3^k} \\ &= \mathbf{D}_0^k\, \mathbf{Q}_{01}^k\, \mathbf{Q}_{12}^k\, \mathbf{Q}_{23}^k \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \end{aligned} \tag{2.25}$$

peut être transformée en un système d'équations linéaires en sinus et cosinus de l'angle θ_2^k de la forme suivante :

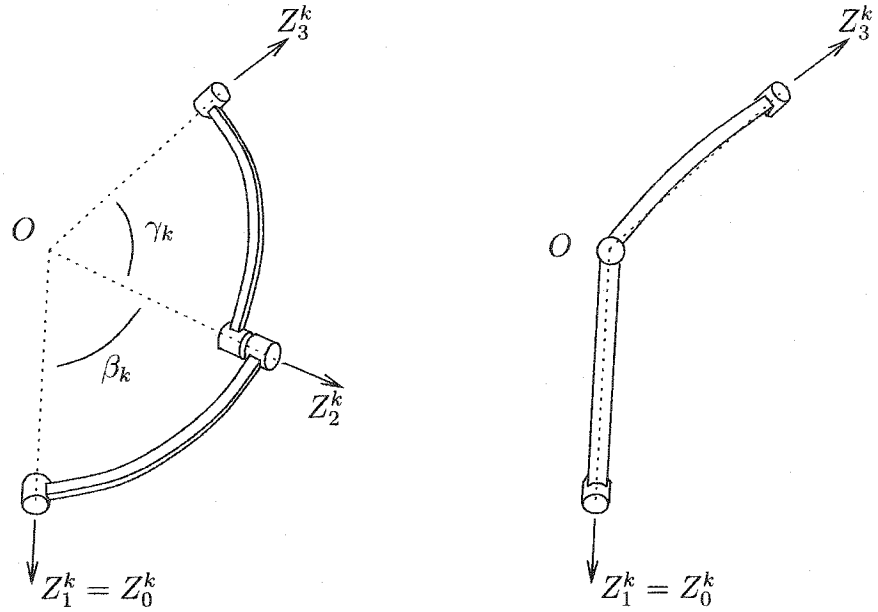
$$\begin{cases} a_0^k \cos\theta_2^k + b_0^k \sin\theta_2^k + c_0^k = 0 \\ a_1^k \cos\theta_2^k + b_1^k \sin\theta_2^k + c_1^k = 0 \\ a_2^k \cos\theta_2^k + b_2^k \sin\theta_2^k + c_2^k = 0 \end{cases} \tag{2.26}$$

Comme ce système d'équation a seulement deux inconnues, $\cos\theta_2^k$ et $\sin\theta_2^k$, mais trois équations, la méthode utilisée pour rendre la résolution plus robuste consiste à prendre l'une des trois équations et de l'ajouter aux deux autres. Les deux équations sont ensuite résolues comme un système traditionnel à deux équations et deux inconnues.

Pour chaque jambe, la possibilité d'avoir deux solutions pour θ_1^k mène à deux couples de solutions (θ_1^k,θ_2^k) qui peuvent être facilement interprétés. En effet, lorsque l'angle $(O,\mathbf{w}_0^k,\mathbf{w}_3^k)$ est égal à la somme $\beta_k+\gamma_k$, la jambe est en pleine extension. C'est le cas où il n'existe qu'une seule solution pour l'angle θ_1^k. Dès que l'angle $(O,\mathbf{w}_0^k,\mathbf{w}_3^k)$ devient plus petit que la somme $\beta_k + \gamma_k$, la jambe doit nécessairement se plier. Comme il y a deux possibilités de pliage, *coude haut* ($\theta_2^k < 0$) et *coude bas* ($\theta_2^k > 0$), il y a alors deux solutions (Figure 2.1).

2.4 Solutions multiples et modes opératoires

Lors de la résolution des équations en sinus et cosinus de θ_1^k (issues de l'équation 2.23 pour $k = A$, B et C), la possibilité d'obtenir deux solutions pour chaque jambe mène à un maximum de huit ensembles de solutions possibles pour une même orientation de l'effecteur. Ceux-ci définissent huit modes opératoires qui ont été numérotés en fonction de la position coude haut ou coude bas de chaque jambe (ta-

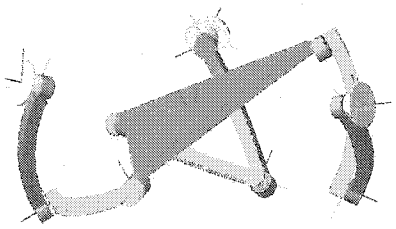


Position *coude bas* Position *coude haut*

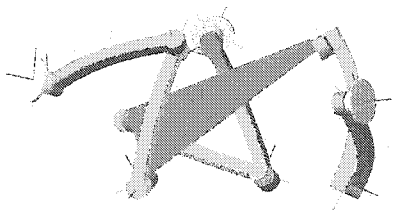
FIGURE 2.1 – Les deux solutions possible pour une jambe du manipulateur

bleau 2.1). La figure 2.2 présente un exemple des huit modes pour une orientation donnée.

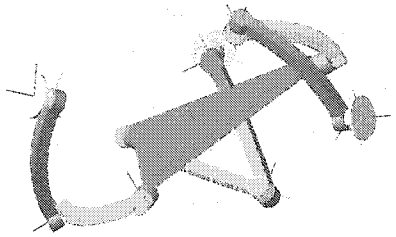
Lorsque le manipulateur est actionné par des moteurs fixés sur le bâti, les angles θ_1^k sont les inconnues importantes à résoudre. Ensuite, les valeurs des angles θ_2^k et θ_3^k, ne sont que secondaires puisque les joints intermédiaires et distaux sont passifs. En pratique, la valeur de θ_2^k peut être très utile pour connaître la position de la jambe k, c'est-à-dire coude haut ou coude bas.



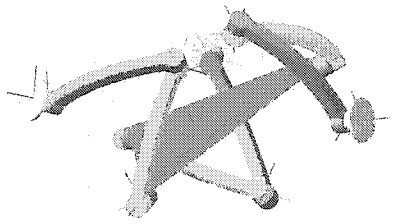
(a) mode 1



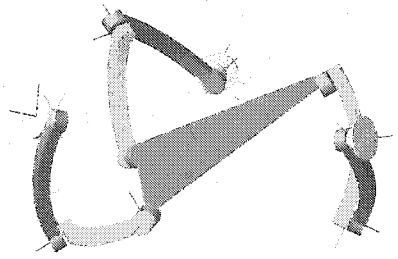
(b) mode 2



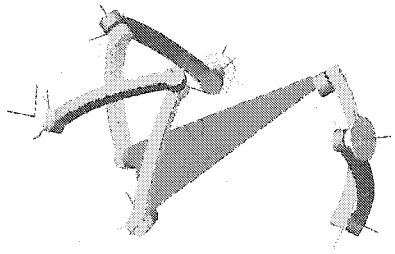
(c) mode 3



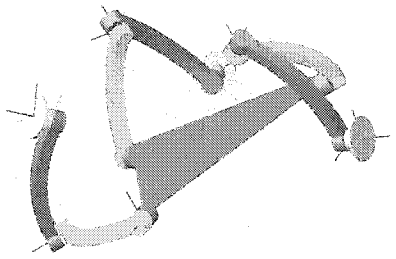
(d) mode 4



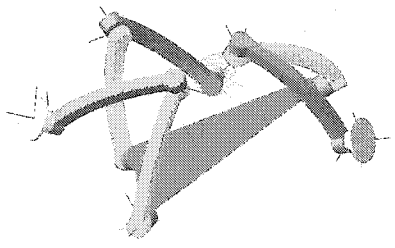
(e) mode 5



(f) mode 6



(g) mode 7



(h) mode 8

FIGURE 2.2 – Les huit modes d'opération

TABLEAU 2.1 – Les huit modes d'opération

Mode	Jambe A	Jambe B	Jambe C
1	Coude bas	Coude bas	Coude bas
2	Coude haut	Coude bas	Coude bas
3	Coude bas	Coude haut	Coude bas
4	Coude haut	Coude haut	Coude bas
5	Coude bas	Coude bas	Coude haut
6	Coude haut	Coude bas	Coude haut
7	Coude bas	Coude haut	Coude haut
8	Coude haut	Coude haut	Coude haut

CHAPITRE 3

CALCUL DES ESPACES DE TRAVAIL AVEC LA MÉTHODE DES 2^K-ARBRES

3.1 Objectifs

Comme expliqué dans l'introduction, la taille restreinte de l'espace de travail des manipulateurs parallèles est une des grandes faiblesses de ce type de manipulateur. C'est pourquoi il est nécessaire d'évaluer au mieux les caractéristiques (taille, forme, volume, etc.) de l'espace de travail en fonction des objectifs de design qui ont été fixés. Dans la plupart des cas, le calcul analytique de l'espace de travail n'est pas faisable car trop complexe. Il faut donc utiliser des méthodes numériques pour le représenter, en acceptant que celui-ci ne sera connu que de manière approximative.

Le principal objectif pour le calcul numérique des espaces de travail est de limiter le temps de calcul nécessaire et l'espace mémoire utilisé au strict minimum. En effet, un temps de calcul trop long rendrait l'optimisation par algorithme génétique laborieuse car beaucoup d'espaces de travail doivent être calculés.

3.2 Les 2^k-arbres sous leur forme d'origine

Les 2^k-arbres sont une structure de données généralisant les arbres binaires (pour $k = 1$, on obtient en fait un arbre binaire). Cette technique est couramment utilisée dans de nombreux travaux de recherche pour calculer les espaces de travail mais aussi pour représenter des volumes (MEAGHER, 1982) ou encore dans la génération

de maillages. Le travail effectué dans ce projet se base principalement sur les travaux de CHABLAT (1998) et TREMBLAY (1999) dans le domaine de la robotique.

Les 2^k-arbres permettent de discrétiser un espace de dimension k en sous-espaces mutuellement exclusifs et de les organiser dans une structure de données de manière déterministe. Les structures les plus connues sont les *quadtrees* et les *octrees*, obtenus respectivement pour $k = 2$ et $k = 3$.

En robotique, la dimension de l'espace dépend généralement du nombre de degrés de libertés du manipulateur étudié. Dans le cas d'un manipulateur plan à deux degrés de libertés, un *quadtree* sera utilisé pour discrétiser un espace plan décrit par deux coordonnées. Il faut toutefois noter que l'espace ne doit pas nécessairement être vectoriel mais les différentes dimensions doivent par contre être indépendantes. Par exemple, dans la famille des manipulateurs parallèles 3-RRR, il est possible de construire des manipulateurs plans à trois degrés de liberté : deux en translation (l'effecteur décrit un espace plan) et un en rotation (l'effecteur peut pivoter sur lui-même). L'espace de travail de ce manipulateur peut être étudié avec un *octree* même si les dimensions ne sont pas de même nature.

L'autre constante importante dans l'utilisation des 2^k-arbres est la profondeur de discrétisation, notée p dans la suite de ce mémoire. La discrétisation consiste à subdiviser en deux parties égales, dans les k directions, un sous-espace donné. Pour un sous-espace donné, la discrétisation génèrera donc 2^k sous-espaces plus petits. Dans le cas d'un *quadtree* et d'un *octree*, le sous-espace est donc subdivisé respectivement en quatre et en huit nouveaux sous-espaces. Les figures 3.1 et 3.2 montrent la manière dont sont discrétisés les *quadtrees* et les *octrees*. À mesure que la profondeur augmente, les sous-espaces obtenus sont à nouveau subdivisés selon le même principe. L'impact de la profondeur sur la discrétisation est présentée de manière visuelle sur la figure 3.1.

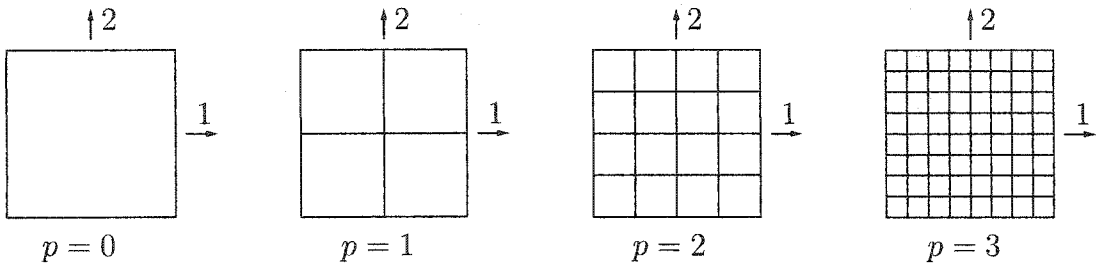


FIGURE 3.1 – Influence de la profondeur p de discrétisation sur un *quadtree*

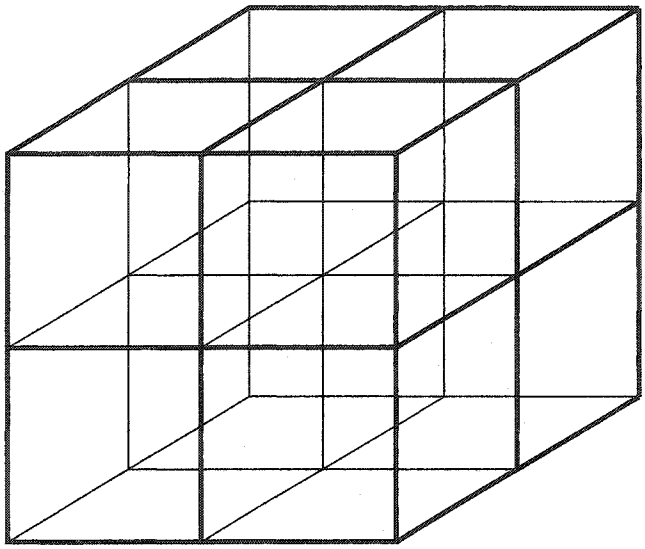


FIGURE 3.2 – Décomposition d'un volume en sous-volumes dans un *octree* de profondeur $p = 1$ (Adapté de CHABLAT (1998))

Le choix d'une valeur adéquate pour la profondeur est crucial car il faut trouver un équilibre en une discrétisation trop grossière ou trop fine. En effet, l'usage des 2^k-arbres en robotique nécessite de faire une hypothèse importante : les propriétés du manipulateur étudiés, telles que l'existence ou non de solutions pour le modèle géométrique inverse, doivent rester quasi-constantes sur la portion de l'espace décrite par l'élément le plus petit. Dans le cas d'une profondeur trop faible, cette hypothèse est loin d'être vérifiée ce qui rend le calcul trop approximatif pour être utilisable. Par contre, dans le cas d'un discrétisation trop grande, le calcul sera certes plus précis mais exigera beaucoup de temps de calcul et d'espace mémoire.

En effet, pour une profondeur et une dimension donnée, le nombre de subdivisions de l'espace est de $2^{k \times p}$. Ajouter un niveau de discrétisation en plus augmente de manière substantielle le nombre de subdivisions.

La structure en arbre est une structure hiérarchisée qui reflète la méthode utilisée pour discrétiser les sous-espaces. Elle exige l'utilisation de *nœuds* ayant chacun un nœud *parent* et des nœuds *enfants*, à l'exception du nœud *racine* qui n'a pas de nœud parent (figure 3.4). À chaque nœud est associée un sous-ensemble de l'espace. La discrétisation de ce sous-espace génère 2^k sous-ensembles et donc 2^k nœuds enfants. La figure 3.4 montre la représentation dans le plan de la structure en arbre de la figure 3.3.

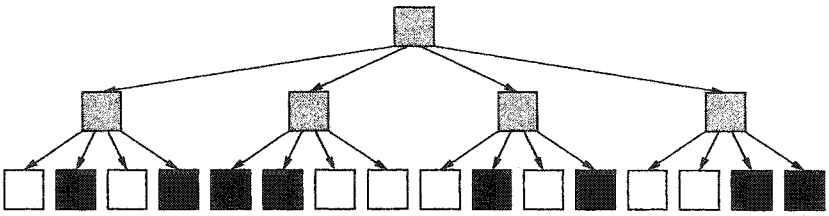


FIGURE 3.3 – Structure d'un *quadtree* de profondeur $p = 2$ (CHABLAT, 1998)

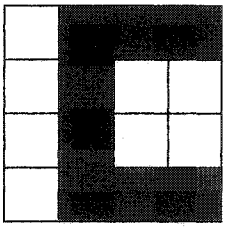


FIGURE 3.4 – Représentation d'un *quadtree* de profondeur $p = 2$ (CHABLAT, 1998)

Afin d'organiser la structure hiérarchique de manière déterministe, une méthode de numérotation permettant d'identifier de manière unique chaque noeud est mise en place (MORTON, 1966). Elle met à profit les caractéristiques de la discrétisation utilisée. En effet, pour chaque direction k, l'espace est subdivisé en deux parties égales. En assignant un bit dans chaque direction, et en empilant les différents bits dans un ordre fixe, chaque sous-espace a donc un numéro d'identification unique.

FIGURE 3.5 – Numérotation des sous-espaces dans un espace à 1 dimension

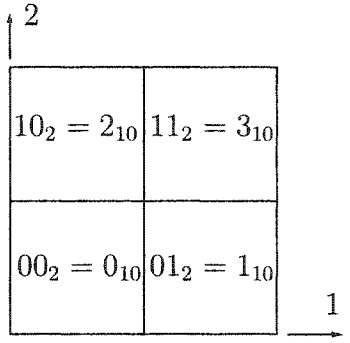


FIGURE 3.6 – Numérotation des sous-espaces dans un *quadtree* (CHABLAT, 1998)

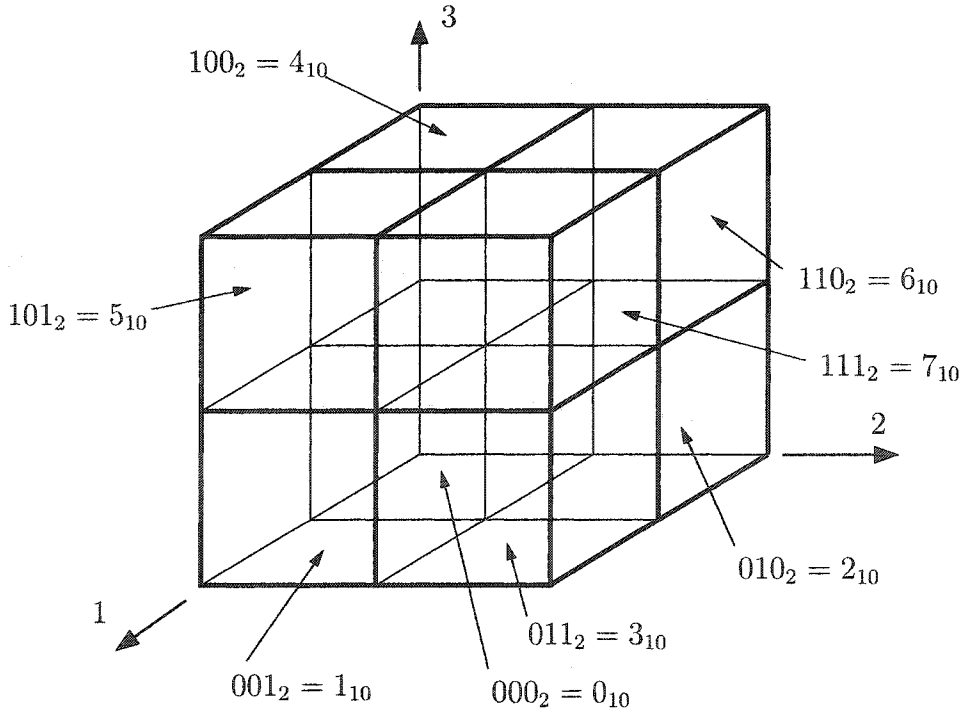


FIGURE 3.7 – Numérotation des sous-espaces dans un *octree*

Par ailleurs, le numéro détermine exactement la position du sous-espace dans son espace parent. Les figures 3.5, 3.6 et 3.7 présentent respectivement la technique de numérotation pour $k = 1$, $k = 2$ et $k = 3$. Il faut noter que la position des bits (de droite à gauche) a été associée au numéro de la direction par rapport à laquelle la subdivision a été réalisée (ce qui explique l'usage de numéros à la place de lettres pour les systèmes d'axes.).

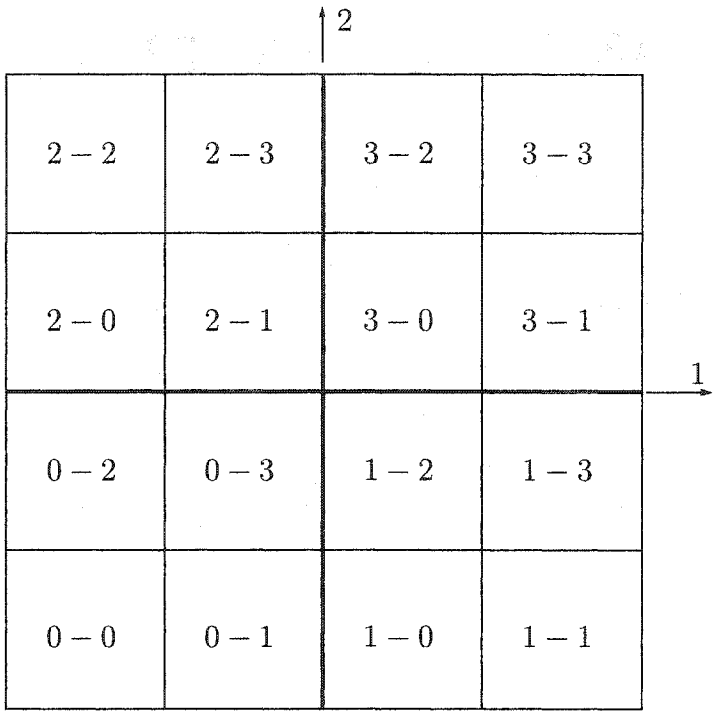


FIGURE 3.8 – Numérotation des nœuds dans un *quadtree* de profondeur $p = 2$ (Adapté de CHABLAT (1998))

Pour pouvoir connaître le sous-espace délimité par un nœud et sa position absolue dans l'espace, il faut parcourir l'arbre à l'envers, c'est-à-dire du nœud recherché jusqu'au nœud racine. À chaque niveau, le numéro associé au nœud courant nous informe de la position de celui-ci par rapport à son nœud parent. En collectant les numéros, il est donc possible de définir un *chemin* unique vers le nœud étudié. Par exemple, dans le cas d'un *quadtree*, le résultat pour un arbre de profondeur 2 est présenté sur la figure 3.8. Il faut noter qu'à mesure que l'on descend dans la

hiérarchie, les numéros sont ajoutés à droite. Pour connaître la taille du sous-espace associé à un nœud donné, il suffit de déterminer à quel niveau de la hiérarchie il est positionné à l'aide du chemin menant à lui.

Dans la thèse de CHABLAT (1998) et le mémoire de TREMBLAY (1999), une couleur est associée à chaque nœud pour déterminer si celui-ci fait partie entièrement (couleur noire) ou partiellement (couleur grise) de l'espace de travail du manipulateur ou n'en fait pas partie du tout (couleur blanche). Ainsi, un nœud ayant tous ses enfants de couleur noire (respectivement blanche ou grise) sera aussi de couleur noire (respectivement blanche ou grise). Un noeud n'ayant que quelques enfants de couleur noire sera de couleur grise.

Pour affecter une couleur à chacun des noeuds, l'espace est parcouru et un critère d'appartenance (tel que l'existence d'une solution pour le modèle géométrique inverse) est appliqué à chaque nœud *de la profondeur maximum*, appelé nœud *feuille* dans ce mémoire. Les couleurs sont ensuite propagées aux noeuds parents.

Dans les travaux de CHABLAT (1998) et de TREMBLAY (1999), les noeuds de couleur blanche sont éliminés de l'arbre pour sauver de l'espace mémoire. Les figures 3.9 et 3.10 illustrent un exemple de résultat. Dans cette optique, le concept de couleur n'est plus utile et peut donc être supprimé dans la représentation de l'arbre. En effet, tous les nœuds feuille sont alors de couleur noire.

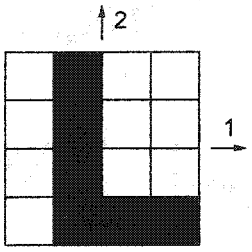


FIGURE 3.9 – Espace de discrétisation de la lettre 'L'

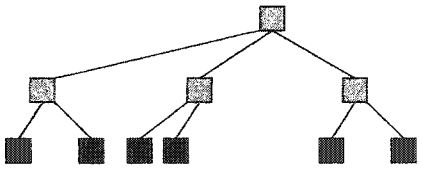


FIGURE 3.10 – *Quadtree* de la lettre 'L' (voir figure 3.9)

3.3 Contribution personnelle — adaptation de la structure d'origine

Dans sa forme d'origine, le nœud d'un 2^k-arbre n'a qu'une couleur associée, selon qu'il appartienne entièrement, en partie ou aucunement à l'espace de travail du manipulateur. Toutefois, beaucoup de paramètres locaux autres que l'existence ou non de solutions pour le modèle géométrique inverse peuvent être intéressants en vue d'une analyse plus poussée du manipulateur. Une liste non exhaustive des paramètres qui pourraient être utiles est présentée ci-dessous :

- le nombre de solutions ou encore les modes opératoires disponibles à une posture donnée du manipulateur,
- l'existence ou non de collisions entre les membrures du manipulateur,
- le degré d'isotropie associé à chaque solution,
- le couple maximal à fournir sur chaque actionneur pour obtenir un effort « de référence » sur l'effecteur.

Les deux premiers paramètres sont couramment utilisés dans ce projet de recherche à des fins de visualisation dans le simulateur logiciel, tel qu'expliqué aux chapitres 4 et 5 et à des fins d'optimisation dans la synthèse géométrique présentée au chapitre 6.

La connaissance de la forme des modes opératoires calculés à l'avance et des collisions possibles entre les membrures pourrait être utile dans la génération de trajectoires pour le manipulateur, afin de choisir quel(s) mode(s) opératoire(s) utiliser

pour avoir une trajectoire optimale du manipulateur. Dans le cas où plusieurs trajectoires seraient possibles, l'usage du degré d'isotropie ou de la puissance à fournir aux actionneurs pourrait servir de critère secondaire.

Il existe toutefois des désavantages à ajouter des informations à l'arbre. D'abord, il n'est plus possible de simplifier l'arbre aussi facilement qu'auparavant. En effet, si tous les nœuds enfants d'un nœud de l'arbre étaient de la même couleur, il était possible de supprimer les nœuds enfants. Que doit-on faire lorsque les autres paramètres locaux stockés dans les noeuds enfants n'ont pas la même valeur? Ensuite, quelle signification faut-il donner aux paramètres dans les nœuds intermédiaires?

La méthode choisie pour éviter ces désavantages serait de ne faire aucune simplification et de ne tenir compte que des nœuds feuilles. Une autre raison vient appuyer ce choix. En utilisant le chemin complet et en l'associant systématiquement aux nœuds, il est tout à fait faisable d'éliminer tous les nœuds intermédiaires car la structure hiérarchique est conservée grâce à la technique de numérotation utilisée. Enfin, en ne conservant que les nœuds feuilles, des gains substantiels en espace mémoire sont réalisés. En effet, un arbre complet de profondeur p et de dimension k contiendrait un nombre de nœud total donné par la formule suivante :

$$\sum_{i=0}^{p} 2^{k\times(p-i)} \tag{3.1}$$

En éliminant tous les nœuds intermédiaires, le nombre de nœuds stockés en mémoire diminue de :

$$\sum_{i=0}^{p-1} 2^{k\times(p-i)} \tag{3.2}$$

Par exemple, avec $k = 3$ et $p = 5$, c'est un gain d'environ 15 % en espace mémoire qui est réalisé.

3.4 Une librairie générique

Afin de ne pas limiter l'usage des 2^k-arbres au calcul exclusif d'espaces de travail (i.e. sans autre paramètre local) ou à l'usage de dimensions 2 ou 3 (i.e. *quadtree* ou *octree*), un effort a été fourni pour créer une librairie en C++ la plus générique possible. Les objectifs fixés étaient les suivants :

- laisser le choix à l'usager de rajouter ou non des paramètres supplémentaires dans les nœuds,
- fournir une librairie qui permette d'utiliser des 2^k-arbres de n'importe quelle profondeur et de n'importe quelle dimension ($k = 1, 2, 3$, et même 4, 5, etc.),
- rendre la librairie compatible avec la librairie standard du C++, entre autre, pour pouvoir utiliser le concept d'itérateur et les algorithmes standard si nécessaires.

De manière pratique, le résultat est d'une incroyable simplicité grâce à la librairie STL (*Standard Template Library*). En effet, programmer un conteneur particulier n'a pas été nécessaire car les propriétés des 2^k-arbres sont identiques avec celles des conteneurs standards `map` et `set` : ce sont des conteneurs ordonnés qui n'acceptent pas de doublons (JOSUTTIS, 1999).

Le cœur de la librairie est un *template* de classe `Path` qui prends comme paramètres la profondeur et la dimension choisies et qui symbolise un chemin vers un nœud feuille. Lorsque aucun paramètre supplémentaire autre que l'existence ou non de solutions pour le modèle géométrique inverse n'est requis, il suffit de stocker les instances de la classe `Path` dans un conteneur `set` pour simuler la construction d'un 2^k-arbre ordinaire. Quand des paramètres supplémentaires doivent être stockés, il suffit de remplacer le conteneur `set` par le conteneur `map`. Celui-ci est un conteneur associatif, c'est-à-dire qu'il permet d'associer une valeur, ou plus généralement un

objet, à une clef (*Key/value pair* en Anglais). La clef correspond en fait au chemin et ne peut être dupliquée au sein du conteneur `map`.

L'autre classe, `Box`, est aussi un *template* de classe prenant les mêmes paramètres que la classe `Path`. Elle permet non seulement de définir l'espace à discrétiser mais aussi de déterminer la position et la taille du sous-espace associé à un nœud feuille. Comme il est possible de déterminer la position et la taille d'un sous-espace associé à un nœud feuille directement en utilisant le chemin vers ce nœud, il n'est pas nécessaire de stocker cette information là dans le conteneur.

Les codes sources de la librairie ainsi que deux exemples d'utilisation sont disponibles à l'annexe IV. Les conditions d'utilisation du logiciel sont présentées dans l'annexe III.

3.4.1 Fonctionnement interne et usage du *template* de classe `Path`

La déclaration du *template* de classe `Path` est la suivante :

```
template< int dim, int depth > class Path
```

où `dim` et `depth` sont respectivement la dimension et la profondeur du 2^k-arbre.

Les méthodes publiques disponibles sont présentées ci-dessous :

- `Path()` : constructeur par défaut;
- `Path( const Path< dim, depth > & p)` : constructeur de copie;
- `const int readAt(const int posn) const` : permet de connaître la position d'un nœud (intermédiaire ou feuille) par rapport à son nœud parent à une profondeur `posn` donnée de l'arbre;
- `void writeAt(const int posn, const int val)` : permet d'écrire la posi-

tion d'un nœud (intermédiaire ou feuille) par rapport à son nœud parent à une profondeur `posn` donnée de l'arbre;

- `const Path getNeighbour(const int dir) const` : génère le chemin vers le nœud voisin dans la direction `dir` (plus de détails dans la section 3.5.1);
- `static const int getDimension()` : retourne la dimension k de l'arbre;
- `static const int getDepth()` : retourne la profondeur de l'arbre;
- `static const Path< dim, depth > getFirstPath()` : retourne le chemin vers le premier nœud feuille possible;
- `static const Path< dim, depth > getLastPath()` : retourne le chemin vers le dernier nœud feuille possible;

Les opérateurs de comparaison `==`, `!=`, `>`, `>=`, `<` et `<=` et l'opérateur d'affectation `=` sont aussi programmés. Ils sont en effet nécessaires au bon fonctionnement du conteneur `set` ou `map`. Enfin, pour faciliter l'usage de la classe, des opérateurs de pré- et post-incrémentation et pré- et post-décrémentation sont aussi disponibles.

De manière pratique, lors du calcul d'un espace de travail, tous les chemins sont utilisés car l'ensemble de l'espace discrétisé doit être testé. Ainsi, il suffit de créer une instance de la classe `Path` qui mène vers le premier nœud feuille possible puis de boucler sur tous les éléments, à l'aide d'un opérateur d'incrémentation, jusqu'au dernier chemin possible. Quand un chemin correspond à un nœud où le modèle géométrique inverse a une solution, il suffit de l'insérer dans le conteneur. Il faut noter que les conteneurs de la STL effectuent systématiquement des copies des objets insérés. Il est donc tout à fait convenable d'utiliser une seule instance de la classe `Path` dans la boucle. En pratique, seulement les méthodes `getFirstPath()`, et `getLastPath()` ainsi que l'opérateur de pré- ou post-incrémentation sont utilisés.

Les autres méthodes publiques sont couramment utilisées par la classe `Path` elle-même et la classe `Box` qui a besoin d'extraire des informations des chemins.

La seule méthode privée importante de la classe `Path` est

```
bool recursiveSearch(Path &p, const int dir, const int posn) const
```

qui est utilisée dans la recherche de voisins (section 3.5.1).

Le fonctionnement interne de la classe `Path` est basé sur la manipulation de bits, correspondant à la numérotation présentée plus haut. Le chemin est stocké sous forme d'un entier non signé `size_t` dont la dimension dépend de l'architecture de l'ordinateur utilisé. À chaque niveau dans l'arbre correspond un nombre de bits égal à la dimension de l'espace discrétisé. L'ordre des bits correspond à l'ordre de numérotation des coordonnées de l'espace, en allant du bit de poids le plus faible pour la première coordonnée jusqu'au bit de poids le plus fort pour la dernière. Les valeurs constituant le chemin sont juxtaposées dans le nombre entier en commençant par le niveau le plus profond pour les bits de poids le plus faible vers le niveau 1 (correspondant au nœud racine) pour les bits de poids le plus fort.

L'usage des classes programmées requiert une attention particulière car aucun mécanisme de vérification des valeurs utilisées n'a été programmé. De plus, un nombre limité de combinaisons profondeur/dimension de l'arbre peut être utilisé car il ne faut pas dépasser le nombre de bits disponibles dans l'entier de type `size_t`. Le nombre de bits utilisés se calcule simplement avec la formule $p \times k$.

3.4.2 Fonctionnement interne et usage du *template* de classe `Box`

La déclaration du *template* de classe `Box` est la suivante :

```
template< int dim, int depth, typename type = float > class Box
```

où `dim` et `depth` sont respectivement la dimension et la profondeur du 2^k-arbre et `type` est le format de nombre à utiliser.

Les méthodes publiques disponibles sont présentées ci-dessous :

- `Box()` : constructeur par défaut;
- `Box( const Box< dim, depth, type > & b)` : constructeur de copie;
- `void setBoundaries( int dir, type min, type max )` : permet de définir les bornes de l'espace à discrétiser dans la direction `dir`;
- `void setMinBoundary( int dir, type min )` : permet de définir la borne inférieure de l'espace à discrétiser dans la direction `dir`;
- `void setMaxBoundary( int dir, type max )` : permet de définir la borne supérieure de l'espace à discrétiser dans la direction `dir`;
- `pair< type, type > getBoundaries( int dir ) const` : renvoie les bornes dans la direction `dir` de l'espace discrétisé;
- `type getSize( int dir ) const` : retourne la longueur des sous-espaces associés aux nœuds feuilles dans la direction `dir`;
- `type getVolume() const` : retourne le volume des sous-espaces associés aux nœuds feuilles;
- `vector<type> getCenterAt( const Path< dim, depth > & p ) const` : retourne les coordonnées du centre du nœud feuille ayant pour chemin `p`;
- `static const int getDimension()` : retourne la dimension k de l'arbre;
- `static const int getDepth()` : retourne la profondeur de l'arbre;

La seule méthode privée, `void computeVolume()`, sert à calculer le volume des sous-espaces associés aux nœuds feuilles dès qu'une des bornes de l'espace à discrétiser est modifiée.

L'usage de la librairie développée ne nécessite en fait aucune discrétisation. En effet, la méthode `getCenterAt()` ne fait que simuler cette discrétisation en calculant la position dans l'espace associé au chemin vers un nœud feuille. L'opération consiste à calculer les coordonnées d'abord sous forme de nombres entiers puis à les transformer en valeur réelles à l'aide des dimensions données de l'espace discrétisé.

3.5 Algorithmes utiles

3.5.1 Recherche des voisins d'un nœud

CHABLAT (1998) propose dans sa thèse une méthode astucieuse pour trouver les nœuds voisins dans les quatre directions (X+, X-, Y+, Y-) d'un *quadtree*. Elle repose sur le calcul du chemin possible dans la direction désirée. La technique décrite maintenant est une généralisation pour les 2^k-arbres de dimension quelconque mais les exemples donnés utilisent un *quadtree* de profondeur 3 dont les chemins aux nœuds feuilles sont présentés à la figure 3.11.

La programmation de l'algorithme de recherche des voisins fait appel à la méthode récursive privée de la classe `Path` nommée :

```
bool recursiveSearch(Path &p, const int dir, const int posn) const
```

et à la manipulation individuelle de bits dans les chemins menant aux nœuds feuilles.

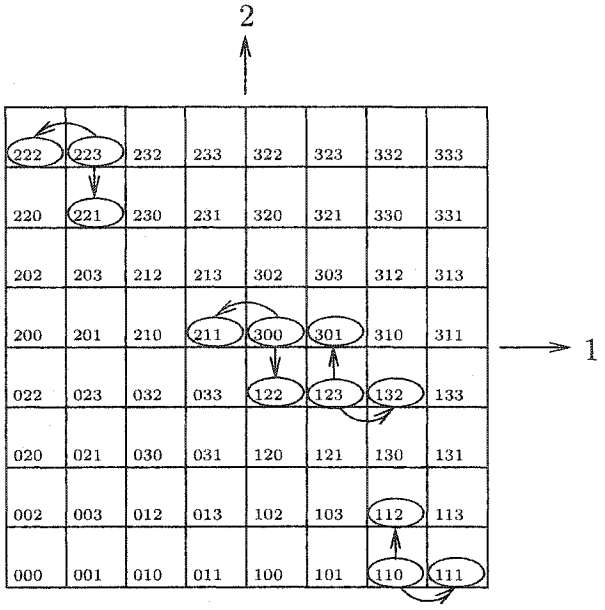


FIGURE 3.11 – Chemins associés aux nœuds feuilles pour un *quadtree* de profondeur $p = 3$

3.5.1.1 Recherche dans une direction positive

Pour toute la procédure de recherche, seul le bit associé à la direction dans laquelle le voisin est recherché est utilisé. Le point de départ est à la profondeur maximale. Si nécessaire, l'algorithme remontera dans la hiérarchie. À une profondeur donnée, l'algorithme récursif consiste à effectuer les opérations suivantes :

- Si, dans la direction de recherche, la valeur qui lui est associée dans la numérotation est nulle, il suffit de lui affecter la valeur 1 pour obtenir le chemin du nœud voisin. La récursion s'arrête alors.
- Si, dans la direction de recherche, la valeur qui lui est associée dans la numérotation est déjà égale à 1, il est obligatoire de remonter dans la hiérarchie :
  - Si la profondeur actuelle dans la hiérarchie est déjà égale à 1, il est impossible de remonter plus haut dans la hiérarchie. Aucun voisin n'a

alors été trouvé dans la direction de recherche. Ceci signifie que le nœud d'origine se trouvait sur le bord de l'espace discrétisé.

- Sinon, avant de remonter d'un étage dans la hiérarchie, il faut affecter la valeur 0 dans la numérotation pour la direction de recherche. Ensuite, il faut répéter l'ensemble des opérations pour le niveau supérieur.

Dans le cas du *quadtree* de profondeur $p = 3$ de la figure 3.11, les voisins du nœud 110 sont recherchés dans les deux directions positives. Comme la valeur associée à la profondeur maximale dans le chemin du nœud étudié est nulle, la recherche s'arrête directement après une récursion. Dans la direction 1, il suffit d'affecter la valeur 1 au bit associé à la direction 1, ce qui mène au chemin 111. Dans le cas de la direction 2, l'affectation de la valeur 1 au bit associé à la direction 2 mène au chemin 112.

Par contre, pour les voisins du nœud ayant pour chemin 123 à la figure 3.11, plusieurs récursions sont nécessaires. Dans la direction 1, le bit associé à cette direction pour la profondeur maximale a déjà la valeur 1. Il suffit de le mettre à zéro et de remonter d'un cran dans la hiérarchie. La numérotation à cette profondeur est 2 ce qui signifie que le bit associé à la direction de recherche a la valeur 0. On lui affecte donc la valeur 1 et la récusion s'arrête. Le chemin du voisin dans la direction 1 positive est donc 132. Dans la direction 2, à la profondeur maximale la numérotation est 3. Le bit associé à cette direction a déjà la valeur 1. Il faut donc le mettre à zéro et remonter dans la hiérarchie. À la profondeur intermédiaire, le bit associé à la direction 2 a aussi la valeur 1, il est donc remis à zéro. En remontant d'un cran dans la hiérarchie, la profondeur minimale est atteinte. Comme le bit associé à la direction 2 a la valeur 0, il faut lui affecter la valeur 1. La récursion s'arrête alors et un voisin dont le chemin est 301 a été trouvé.

3.5.1.2 Recherche dans une direction négative

La procédure de recherche dans une direction négative est similaire à celle décrite ci-dessus. La profondeur de départ est toujours la profondeur maximale de la hiérarchie. À une profondeur donnée, l'algorithme récursif s'exécute de la manière suivante :

- Si, dans la direction de recherche, la valeur qui lui est associée dans la numérotation est égale à 1, il suffit de lui affecter la valeur 0 pour obtenir le chemin du nœud voisin. La récursion s'arrête alors.
- Si, dans la direction de recherche, la valeur qui lui est associée dans la numérotation est déjà nulle, il est obligatoire de remonter dans la hiérarchie :
  - Si la profondeur actuelle dans la hiérarchie est déjà égale à 1, il est impossible de remonter plus haut dans la hiérarchie. Aucun voisin n'a alors été trouvé dans la direction de recherche. Ceci signifie que le nœud d'origine se trouvait sur le bord de l'espace discrétisé.
  - Sinon, avant de remonter d'un étage dans la hiérarchie, il faut affecter la valeur 1 dans la numérotation pour la direction de recherche. Ensuite, il faut répéter l'ensemble des opérations pour le niveau supérieur.

Ainsi, les voisins du nœud ayant pour chemin 223, à la figure 3.11, dans les directions 1 et 2 négatives ont respectivement pour chemin 222 et 221. Ils s'obtiennent en une seule récusion. Par contre, dans le cas du nœud ayant pour chemin 300, il faut remonter deux fois dans la hiérarchie. Les chemins des nœuds voisins dans les directions 1 et 2 négatives sont respectivement 211 et 122.

3.5.2 Algorithme d'élimination des facettes cachées

Pour la visualisation des *octrees*, la librairie graphique *OpenGL* a été utilisée. Dans un espace cartésien, chaque nœud feuille correspond à une boîte de forme rectangulaire. Il y a donc six faces à afficher dont plusieurs peuvent être complètement cachées à l'intérieur du volume, s'il existe plusieurs boîtes adjacentes. L'algorithme proposé permet d'éliminer ces facettes afin que l'affichage soit grandement accéléré et que les ressources en mémoire ne soient pas monopolisées inutilement. Il faut noter que CHABLAT (1998) utilise une méthode plus complexe basée sur le regroupement des boîtes adjacentes en boîtes de plus grandes dimensions sans éliminer les facettes communes.

L'algorithme d'élimination des facettes cachées met à profit le conteneur `map` de la librairie standard STL. Tout comme pour les 2^k-arbres, la clé utilisée est en fait le chemin du nœud feuille, c'est-à-dire l'instance de la classe `Path` associée au nœud feuille. L'objet associé à la clé est une classe `NodeFlags` permettant de stocker des *drapeaux* pour connaître quelles facettes de la boîte englobante seront affichées par la suite.

L'élimination des facettes se déroule comme suit. Tous les nœuds feuilles de l'espace de travail sont parcourus. Pour chacun d'eux, il faut rechercher s'il a des voisins présents dans les trois directions (plus précisément, seulement dans les directions négatives, sinon des tests seraient effectués deux fois.). Dans le cas où un voisin est trouvé, les *drapeaux* pour les facettes communes au nœud et à son voisin sont affectés de la valeur « facette cachée ». Lors de l'affichage, pour chaque nœud, les *drapeaux* sont testés pour ne générer que les facettes restantes.

3.6 Application au manipulateur sphérique 3-RRR

L'espace de travail du manipulateur est calculé dans l'espace des angles d'EULER Z-X-Z ψ_1, ψ_2 et ψ_3 à l'aide du modèle géométrique inverse. Afin de faciliter l'analyse des résultats, pour chaque orientation de l'effecteur, les informations stockées dans l'*octree* sont toutes les solutions regroupées par mode opératoire ainsi que la présence ou non de collisions pour chaque solution.

3.6.1 Forme de l'espace discrétisé

Afin de couvrir l'ensemble des orientations possibles à l'aide des angles d'EULER, il faut trouver les intervalles correspondants pour chacune des coordonnées. La figure 3.12 représente les deux premiers angles d'EULER décrivant une sphère.

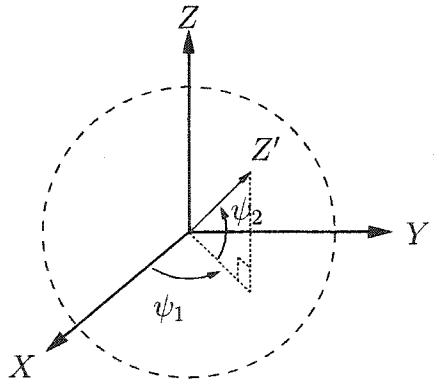


FIGURE 3.12 – La sphère décrite par les deux premiers angles d'EULER

La sphère est complètement décrite lorsqu'un des deux angles varie entre $-\pi$ et π tandis que l'autre varie entre $-\frac{\pi}{2}$ et $\frac{\pi}{2}$. À chaque position donnée par (ψ_1,ψ_2), l'angle ψ_3 peut faire pivoter le repère de 2π sans risque d'avoir plusieurs coordonnées décrivant la même orientation. Ainsi, dans le cadre de ce mémoire, il a été décidé

que :

$$\psi_1 \in]-\frac{\pi}{2};\frac{\pi}{2}] \tag{3.3a}$$

$$\psi_2 \in]-\pi;\pi] \tag{3.3b}$$

$$\psi_3 \in]-\pi;\pi] \tag{3.3c}$$

La forme de l'espace discrétisé à l'aide de l'*octree* est donc une boîte rectangulaire, lorsque les trois angles d'EULER sont représentés dans un repère cartésien. Les limites de la boîte correspondent aux intervalles présentés ci-dessus. Les singularités mathématiques dues aux angles d'EULER sont évitées car le modèle géométrique inverse n'est calculé qu'au centre de chaque boîte associée aux nœuds feuilles. Le fait que l'espace discrétisé soit centré à l'origine élimine donc toute possibilité d'obtenir des nœuds feuilles dont le centre est sur le plan $\psi_2 = 0$.

3.6.2 Projection sur un espace à deux dimensions

La représentation de l'espace de travail en trois dimensions ne permet pas d'effectuer une analyse qualitative efficace. En effet, représenter l'espace selon les trois angles d'EULER ψ_1, ψ_2 et ψ_3 dans un repère cartésien ne permet pas de s'imaginer facilement quelle est l'orientation de l'effecteur. Une projection sur l'espace à deux dimensions (ψ_1, ψ_2) s'avère facilement visualisable sous la forme d'une portion de sphère, directement dessinée autour du manipulateur. En effet, Les deux angles ψ_1 et ψ_2 servent à désigner dans quelle direction pointe l'effecteur tandis que le dernier, ψ_3, est nécessaire pour faire pivoter l'effecteur autour de l'axe associé à cette direction. Chaque portion infinitésimale de sphère de l'espace de travail projeté correspond à une orientation (ψ_1, ψ_2) de l'effecteur pour laquelle il existe des valeurs de ψ_3 menant à des solutions valides au modèle géométrique inverse. De manière

pratique, lorsque le vecteur unitaire $\mathbf{w}_3$ de la base $\mathcal{B}_3$ rattachée à l'effecteur pointe vers cette portion infinitésimale, au moins une valeur de l'angle ψ_3 dans l'intervalle $]-\pi;\pi]$ devrait mener à une orientation valide de l'effecteur.

Afin de réduire les pertes d'informations dues à la projection sur un espace de dimension inférieure, pour chacune des positions (ψ_1, ψ_2) atteignables par le manipulateur, la proportion d'orientations possibles selon ψ_3 est codée sous la forme d'un pourcentage et affichée à l'aide d'une échelle de couleur. Sur l'exemple de la figure 3.13, la proportion des orientations atteignables atteint environ 47 % en faisant varier ψ_3 pour des valeurs de ψ_1 et ψ_2 constantes. Cette valeur correspondrait à une couleur verte en utilisant l'échelle présentée à la figure 3.17. La portion infinitésimale de sphère de l'espace projeté correspondant à l'orientation (ψ_1, ψ_2) donnée prendra alors la couleur verte.

La technique de projection consiste à prendre chaque chemin d'un nœud feuille de l'espace de travail initial et à trouver son chemin équivalent dans l'espace à deux dimensions. L'espace à deux dimensions (ψ_1, ψ_2) est discrétisé à la même profondeur à l'aide d'un *quadtree*. Les bornes utilisées pour le *quadtree* correspondent exactement aux bornes de l'*octree* pour ψ_1 et ψ_2. Ainsi, pour chaque chemin d'un nœud feuille de l'espace d'origine, il suffit d'éliminer la composante selon ψ_3 pour obtenir un chemin dans l'espace projeté. De manière pratique, le *quadtree* est un conteneur `map` dans lequel est associé à chaque nœud un nombre virgule flottante `double`,

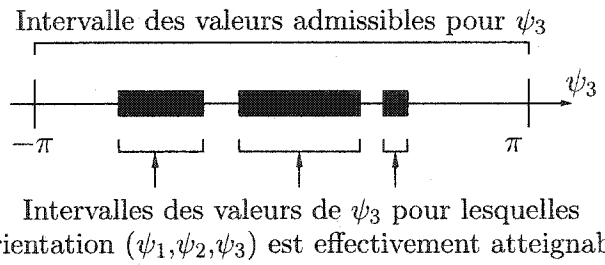


FIGURE 3.13 – Exemple d'orientations atteignables en faisant varier ψ_3 pour ψ_1 et ψ_2 constants.

correspondant au pourcentage atteignable de l'intervalle $]-\pi;\pi]$ pour l'angle ψ_3. Comme plusieurs chemins sources peuvent avoir un chemin commun dans l'espace projeté, la valeur du pourcentage atteignable est incrémentée en conséquence à mesure que l'*octree* de l'espace initial est parcouru.

3.6.3 Un exemple en images

Afin d'illustrer les concepts des paragraphes précédents, un calcul d'espace de travail et une projection sur l'espace à deux dimensions (ψ_1, ψ_2) sont maintenant présentés. Le manipulateur de la figure 3.14 est complètement symétrique. Les angles de la base sont tous égaux à $\alpha_{AB} = \alpha_{AC} = \alpha_{BC} = 115°$ tandis que pour l'effecteur ils prennent la valeur $\delta_{AB} = \delta_{AC} = \delta_{BC} = 120°$. Les trois jambes, identiques, ont une longueur totale de 110° répartie uniformément entre les membrures proximale et distale, c'est-à-dire que $\beta_k = \gamma_k = \frac{110°}{2}$ pour $k = A$, B, et C.

L'espace de travail calculé pour une profondeur de 6 est présenté sur la figure 3.15. La zone discrétisée est délimitée par la boîte aux contours gris. La projection sur l'espace à deux dimensions (ψ_1, ψ_2) est représenté superposé au manipulateur sur la figure 3.16. La couleur de la zone sphérique correspond à un pourcentage atteignable de l'intervalle $]-\pi;\pi]$ pour l'angle ψ_3 à la direction pointée par l'axe Z de l'effecteur.

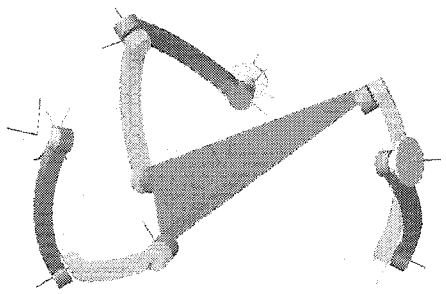


FIGURE 3.14 – Un exemple de manipulateur symétrique

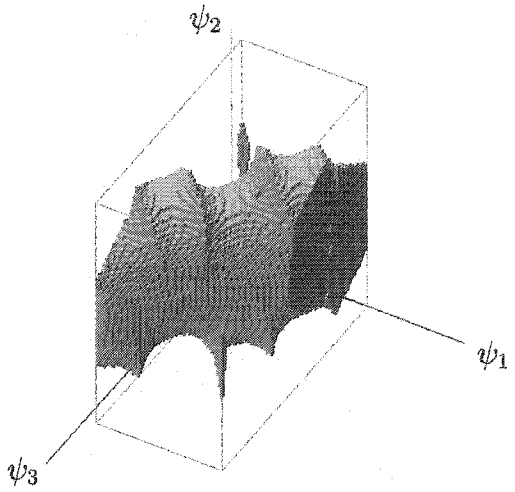


FIGURE 3.15 – L'espace de travail représenté dans un espace cartésien

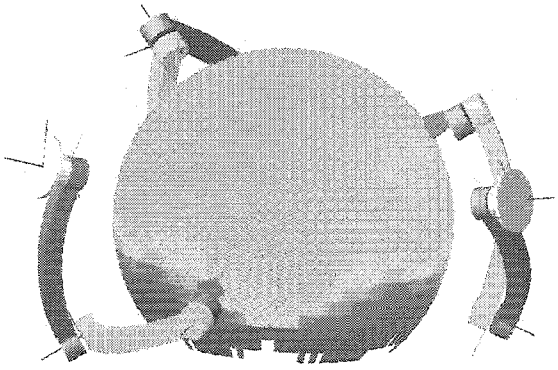


FIGURE 3.16 – L'espace de travail projeté

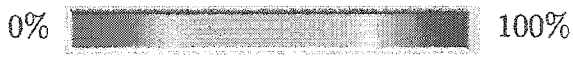


FIGURE 3.17 – L'échelle de couleur utilisée

CHAPITRE 4

TECHNIQUES DE DÉTECTION DE COLLISION

4.1 Présentation générale

Beaucoup de techniques de détection de collision ont été développées ces dernières années. Un des objectifs premiers des recherches effectuées dans ce domaine est d'obtenir des algorithmes peu gourmands en ressources matérielles, tant au niveau de la mémoire que du temps de calcul. La solution idéale serait d'utiliser directement les équations mathématiques des courbes et surfaces utilisées dans le modèle 3D dessiné sur un logiciel de CAO. Mais un calcul de distance sur ces éléments est un problème compliqué à résoudre et demandant un temps de calcul non négligeable, ce qui rend la technique inutilisable pour des applications en temps réel.

Parmi toutes les techniques existantes, celle qui demeure la plus précise tout en étant efficace est celle utilisant des modèles polyédriques convexes. En effet, grâce à l'algorithme de LIN-CANNY (1991), un calcul de distance peut se faire très rapidement. Il existe néanmoins beaucoup d'autres méthodes. Une analyse des différentes techniques est présentée dans un article de LIN et GOTTSCHALK (1998). Parmi celles-ci, on peut citer les boîtes englobantes orientées, *Oriented bounding boxes* (GOTTSCHALK, LIN et MANOCHA, 1996), qui sont à l'origine de la technique filaire développée dans ce mémoire.

Comme la majorité des solides représentés par des modèles polyédriques ne sont pas convexes, il est indispensable de trouver une technique de décomposition en

polyèdres ou surfaces convexes (CHAZELLE et coll., 1995) afin de pouvoir utiliser des algorithmes tels que celui de LIN-CANNY. Il existe toutefois une méthode développée par CARRETERO, NAHON et MA (2002) qui permet d'éviter cette étape complexe. Elle consiste à utiliser des algorithmes génétiques pour déterminer la distance minimale entre deux objets concaves.

4.2 Détection de collision appliquée à un modèle polyédrique

Tous les logiciels de CAO sont obligés d'utiliser des modèles basés sur des triangles ou des rectangles pour l'affichage. En effet, les cartes graphiques sont conçues pour travailler avec des triangles et des rectangles pour effectuer les rendus réalistes d'objets en 3D. Il existe plusieurs fichiers d'échange de données basés sur ces modèles facettisés qui approchent la forme réelle décrite par les équations mathématiques : le VRML, langage balisé pour la réalité virtuelle, et le STL, fichier d'échange pour la stéréolithographie.

Une technique de détection de collision utilisant de tels fichiers est donc tout à fait appropriée car n'importe quel logiciel de CAO est utilisable pour générer des modèles 3D facettisés approximant convenablement les formes réelles du solide. C'est la raison principale qui a guidé le choix de cette première méthode.

Pour décrire un solide avec des facettes triangulaires ou rectangulaires, il est indispensable de savoir de quel côté de chaque triangle la matière se trouve. Il faut donc associer une orientation aux triangles et avoir un polyèdre complètement fermé. Des algorithmes rapides et robustes sont disponibles pour calculer la distance entre deux polyèdres convexes comme, par exemple, celui de LIN-CANNY (1991). Malheureusement, la plupart des modèles polyédriques créés sont non-convexes et une technique de conversion doit alors être utilisée pour mettre à profit ces algorithmes

performants.

Dans le cadre de ce projet, une technique développée à l'Université de Caroline du Nord et disponible sous la forme d'une librairie C++ (nommée SWIFT++) a été utilisée (EHMANN et LIN, 2001). Elle est basée sur une décomposition du modèle polyédrique en surfaces convexes et sur une structure de données hiérarchique de polyèdres convexes.

La première étape consiste à décomposer le modèle polyédrique en plusieurs surfaces convexes. L'algorithme utilisé pour cette tâche consiste à parcourir un graphe contenant les sommets et les côtés de chaque triangle pour chercher des ensembles de triangles connectés et orientés de manière à former une surface convexe. Ensuite, la structure de données hiérarchique est construite en séparant les différentes surfaces de manière itérative et en créant à chaque étape un polyèdre convexe associé au sous-ensemble de surfaces en question. De manière pratique, cet algorithme mène à un ensemble de polyèdres convexes parmi lesquels celui correspondant à la racine de la structure englobe le modèle au complet de manière très imprécise. À mesure que la structure est parcourue, les polyèdres sont plus nombreux, plus petits et correspondent de mieux en mieux au modèle initial. Lors de la construction des polyèdres convexes, un attribut est associé à chacune des facettes. Trois états sont possibles : la facette appartient au modèle initial, la facette a été générée et se trouve en dehors du modèle ou la facette a été générée et se trouve à l'intérieur du volume décrit par le modèle.

La figure 4.2 représente la décomposition en polyèdres convexes d'un volume non convexe présenté à la figure 4.1. En pratique, la décomposition de ce volume ne donnerait pas tout à fait ce résultat en raison des erreurs numériques introduites dans les calculs. En effet, l'algorithme de décomposition en surfaces convexes ne peut gérer correctement des facettes adjacentes disposées sur un même plan.

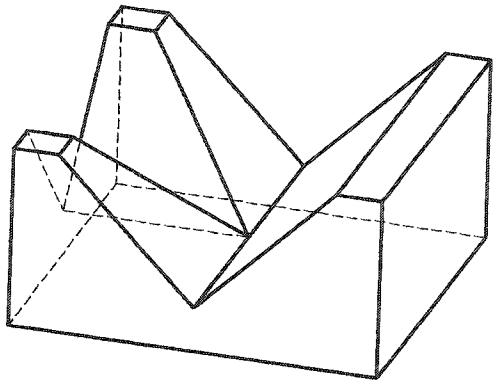


FIGURE 4.1 – Exemple de volume non convexe.

L'intérêt d'utiliser cette hiérarchie est d'accélérer le calcul de collision ou de distance. Il est en effet rapide de déterminer de façon itérative si deux objets s'intersectent en allant chercher au besoin une description plus détaillée de la géométrie à l'intérieur des structures hiérarchiques. Par exemple, si les deux polyèdres de la racine des arbres ne s'intersectent pas, il n'est pas nécessaire d'utiliser les polyèdres plus nombreux et contenant plus de facettes pour savoir que les deux objets ne s'intersectent pas. Par ailleurs, si les deux polyèdres s'intersectent et que les triangles impliqués appartiennent aux modèles initiaux ou sont inclus dans les volumes initiaux, alors il n'est pas non plus nécessaire de chercher plus loin car les objets s'intersectent. Tant qu'il n'est pas possible d'en arriver à cette conclusion, c'est à dire lorsque les polyèdres s'intersectent sur une ou deux facettes situées à l'extérieur des volumes, il est nécessaire d'avancer dans les structures hiérarchiques pour tester les polyèdres convexes approchant plus précisément le modèle initial.

Ainsi, bien que la quantité de mémoire nécessaire pour stocker les structures hiérarchiques soit importante, un gain appréciable de vitesse est réalisé car un nombre limité de calculs est nécessaire pour vérifier la collision entre deux objets. L'intérêt d'utiliser une technique rapide de détection de collision est important dans le cadre

de calculs d'espaces de travail car beaucoup de positions différentes doivent être testées.

4.3 Détection de collision sur géométrie simplifiée (modèle filaire)

Lorsque les objets sont très élancés, comme par exemple des tiges droites ou courbées, un modèle hiérarchique peut rapidement devenir très volumineux. D'autres techniques permettent d'approcher plus grossièrement la géométrie pour améliorer l'usage des ressources matérielles tout en ne compromettant pas trop la précision du résultat. La plus simple consiste à effectuer un modèle filaire de l'objet comme si c'était un maillage d'éléments-poutre en éléments finis (voir fig. 4.3). À chaque segment est associée une épaisseur approchant les dimensions de l'objet (fig. 4.4).

La distance entre deux objets correspond alors à calculer la distance entre les deux segments les plus rapprochés, qui est un calcul géométrique rapide. Lorsque la distance minimale est inférieure aux dimensions associées aux segments impliqués, il y a alors collision entre les deux objets. Cette technique a été développée car elle est extrêmement simple. De plus, il n'est plus nécessaire de créer la géométrie dans un logiciel de CAO grâce à la simplicité du modèle.

Bien qu'extrêmement simple, la technique peut être améliorée afin de limiter les calculs et donc d'accélérer le processus de détection de collision. En effet, si deux solides sont chacun discrétisés à l'aide de trois segments, la recherche de collisions oblige à faire 9 calculs de distance entre segments, même si aucune collision n'est trouvée. En s'inspirant de la technique hiérarchisée développée pour les modèles polyédriques, une méthode semblable a été mise en place pour les modèles filiaires.

Le but de la décomposition hiérarchique est de réduire le nombre de calculs de distance au strict minimum. Le principe de la décomposition proposée est d'avoir

un modèle filaire très grossier à la racine de la hiérarchie et qui devient de plus en plus précis lorsque celle-ci est parcourue. Ainsi, dans le cas où il n'y a aucune collision entre deux modèles filaires, le nombre de calculs de distance peut être réduit à un seul dans le meilleur des cas. Dans la pratique, l'utilisation d'un modèle hiérarchique a permis de diviser par huit le temps de calcul d'un espace de travail avec détection de collision par rapport au calcul avec un modèle simple !

4.3.1 Algorithme de décomposition hiérarchique

Afin d'illustrer de manière visuelle la décomposition hiérarchique d'un modèle filaire, un exemple en deux dimensions est donné à la figure 4.5. La matière entourant chaque segment du solide est montrée en pointillés.

La décomposition hiérarchique de l'exemple de la figure 4.5 est présenté sur la figure 4.6. À la racine de l'arbre binaire se trouve un segment $[P_0P_3]$ dont l'épaisseur est calculé de manière à englober toute la matière du solide discrétisé. À un niveau plus bas, un nœud feuille est atteint dans lequel se trouve le segment $[P_0P_1]$ dont le diamètre d'origine d_0 a été conservé. L'autre nœud comprend un segment $[P_1P_3]$ englobant les deux segments originaux restants. Au dernier niveau de la hiérarchie se trouvent les deux segments originaux $[P_1P_2]$ et $[P_2P_3]$ avec leurs diamètres respectifs d_1 et d_2.

De manière visuelle, il est possible de confirmer que la décomposition hiérarchique proposée n'est pas optimale. Elle a toutefois le mérite d'être extrêmement simple. En effet, elle n'utilise que les sommets existants et le calcul du diamètre associé aux segments fictifs s'effectue très facilement. Soit un segment fictif $[P_iP_j]$. Chaque sommet P_k, avec $k = i+1$ à $j-1$, est projeté orthogonalement en un point P'_k sur le plan passant par le point P_i et de normale $\overrightarrow{P_iP_j}$. Le rayon de matière autour du

segment fictif est la plus grande distance P'_kP_i, ajustée en tenant compte du rayon de la matière autour des segments réels.

4.3.2 Détection de collision entre deux modèles hiérarchiques

Une requête de détection de collision entre deux solides décrits par un modèle filaire hiérarchique est basée sur une opération récursive. Celle-ci consiste en un test entre deux segments de la hiérarchie. Le résultat du calcul de distance entre les segments détermine s'il est nécessaire de chercher ou non une description plus détaillée du solide, c'est-à-dire de descendre d'un niveau dans la hiérarchie. Après calcul, les conditions testées sont les suivantes :

- Si les deux segments testés ne sont pas en collision (i.e. la distance entre les deux est plus grande que la somme des rayons associés aux segments), la fonction retourne l'absence de collision.
- Si les deux segments sont en collision, il y a alors deux cas possibles :
  - si les deux segments sont des nœuds feuilles de la hiérarchie, c'est-à-dire des segments appartenant physiquement au modèle filaire, il y a alors clairement collision entre les deux solides et la fonction retourne cette information.
  - si l'un des deux segments est un nœud intermédiaire de la hiérarchie il est alors nécessaire de descendre d'un niveau dans la hiérarchie concernée et de faire deux nouveaux tests, c'est-à-dire d'appeler à nouveau la fonction récursive pour les deux couples de segments à tester.
  - si les deux segments sont des nœuds intermédiaires, il y a alors quatre tests supplémentaires à réaliser donc quatre appels de la fonction récursive.

Le démarrage de la requête s'effectue en appelant la fonction récursive sur les deux nœuds racines des deux hiérarchies. À cause de la récursivité, la fonction doit propager vers la source de la requête les résultats des appels subséquents de la fonction de test. La figure 4.7 illustre sous forme d'organigramme le fonctionnement de la fonction récursive de détection de collision.

4.4 Avantages et inconvénients des deux techniques — étude qualitative

Le choix d'une méthode plutôt qu'une autre doit être réalisé selon les objectifs à atteindre. Les deux méthodes ont leurs forces et leurs faiblesses qui sont synthétisées au tableau 4.1.

Un modèle filaire est rapide à mettre en place mais ne décrit pas de manière très précise le solide. Ce type de modèle sera donc à privilégier lors de l'étude préliminaire d'un manipulateur, c'est-à-dire lorsque la géométrie définitive n'est pas encore connue.

L'usage d'un modèle polyédrique est donc recommandé lorsque les dimensions définitives du manipulateur sont connues et qu'il est alors nécessaire d'obtenir une grande précision. Le modèle solide issu d'un logiciel de CAO est alors converti en un polyèdre utilisable par la librairie de détection de collision.

Dans le cadre de ce mémoire, l'usage exclusif d'un modèle polyédrique n'aurait pas permis de réaliser une synthèse géométrique du manipulateur. En effet, le lien entre le logiciel d'optimisation et le logiciel de CAO nécessite une intervention humaine, à moins d'avoir l'opportunité de programmer un module d'interface entre les deux. À chaque changement de paramètre géométrique, il est nécessaire de changer manuellement les paramètres puis de régénérer les modèles polyédriques.

TABLEAU 4.1 – Avantages et inconvénients des deux types de modèle utilisés.

Modèle	Avantages	Inconvénients
polyédrique	– précision du modèle – rapidité	– nécessité d'utiliser un logiciel de CAO pour générer le modèle – algorithme de détection de collision plus complexe – difficulté d'adapter le modèle "à la volée"
filaire	– simplicité du modèle – simplicité de l'algorithme de détection de collision – facilité à modifier le modèle "à la volée" – décrit convenablement les solides élancés tels que les bras de robots – rapidité	– description relativement grossière du modèle – les pièces moins élancées sont moins bien décrites

C'est dans ce contexte que l'usage d'un modèle filaire est efficace. Par la simplicité de la représentation, il est possible de régénérer rapidement et automatiquement un modèle dès qu'un paramètre géométrique du manipulateur est modifié.

4.5 Comparaison des deux méthodes dans le calcul d'espaces de travail du manipulateur étudié

Les mécanismes de détection de collision sont rajoutés dans le calcul d'espace de travail sous forme d'une structure rattachée à chaque nœud feuille de l'*octree* conte-

nant toutes les solutions, classées par modes opératoires, ainsi qu'un indicateur des solutions menant à des collisions.

Afin de comparer les résultats obtenus avec les deux techniques de détection de collision et sans détection de collision, deux manipulateurs symétriques particuliers sont présentés. Tous les angles du bâti (α_{kl}) et de l'effecteur (δ_{kl}) font 120°. Toutes les jambes sont identiques. Les membrures proximales et distales ont le même angle. Le premier manipulateur a des membrures de dimensions $\beta_k = \gamma_k = 65°$ tandis que pour le second, $\beta_k = \gamma_k = 90°$.

Pour l'utilisation de modèles polyédriques, les solides ont été modélisés sous CATIA V5 puis convertis en format STL. Un logiciel a ensuite été programmé pour convertir les fichiers STL en un format de fichier compréhensible par la librairie SWIFT++. Un logiciel est fourni avec la librairie pour générer la hiérarchie de polyèdres qui est stockée dans un fichier directement utilisable pour les requêtes de détection de collision.

Pour les modèles filaires, les dimensions des membrures sont choisies pour se rapprocher le plus possible des dimensions des modèles réalisés sous CATIA. Chaque membrure est découpée en trois segments, ce qui s'est avéré un bon compromis entre précision et performance.

Les tableaux 4.2 et 4.3 représentent les pourcentages atteignables de l'espace total décrit par les angles d'EULER selon les différents modes opératoires. Le manipulateur ayant des membrures de 90° est théoriquement le manipulateur parfait car il permet d'atteindre toutes les orientations possibles. En pratique, à cause de la grande dimension des membrures, il existe beaucoup trop de collisions. Un résultat important apparaît alors : pour le manipulateur étudié, l'erreur commise en négligeant les collisions atteint presque 30% dans le pire des cas.

Le calcul d'espaces de travail utilisant la détection de collision avec l'une ou l'autre technique présentée donne des résultats très similaires. Dans le cas d'un discrétisation relativement grossières de l'espace décrit par les angles d'EULER, cette différence devient négligeable.

La rapidité de calcul est environ du même ordre de grandeur avec la librairie SWIFT++ et l'algorithme hiérarchique développé pour le modèle filaire, avec un léger avantage pour ce dernier dans certains cas.

TABLEAU 4.2 – Espace de travail en % de l'espace d'orientation maximal – résultats comparatifs pour $\beta = \gamma = 65°$

Mode d'opération	**Sans détection de collision**	**Modèle polyédrique**	**Modèle filaire**
1 et 8	56,1 %	40,2 %	42,2 %
2 et 7	56,1 %	47,5 %	48,6 %
3, 4, 5 et 6	56,1 %	51,0 %	51,6 %

TABLEAU 4.3 – Espace de travail en % de l'espace d'orientation maximal – résultats comparatifs pour $\beta = \gamma = 90°$

Mode d'opération	**Sans détection de collision**	**Modèle polyédrique**	**Modèle filaire**
1 et 8	100 %	52,9 %	57,1 %
2 et 7	100 %	70,5 %	73,4 %
3, 4, 5 et 6	100 %	71,2 %	73,7 %

4.6 Conclusion

Les résultats présentés dans cette section montrent à quel point la prise en compte des obstructions est indispensable lors du calcul des espaces de travail de manipulateurs parallèles. En effet, dans le cas présent, l'erreur réalisée en négligeant les collisions entre les membrures atteint presque 30%! Par ailleurs, les algorithmes de détection de collision sont simples à mettre en œuvre et relativement rapides.

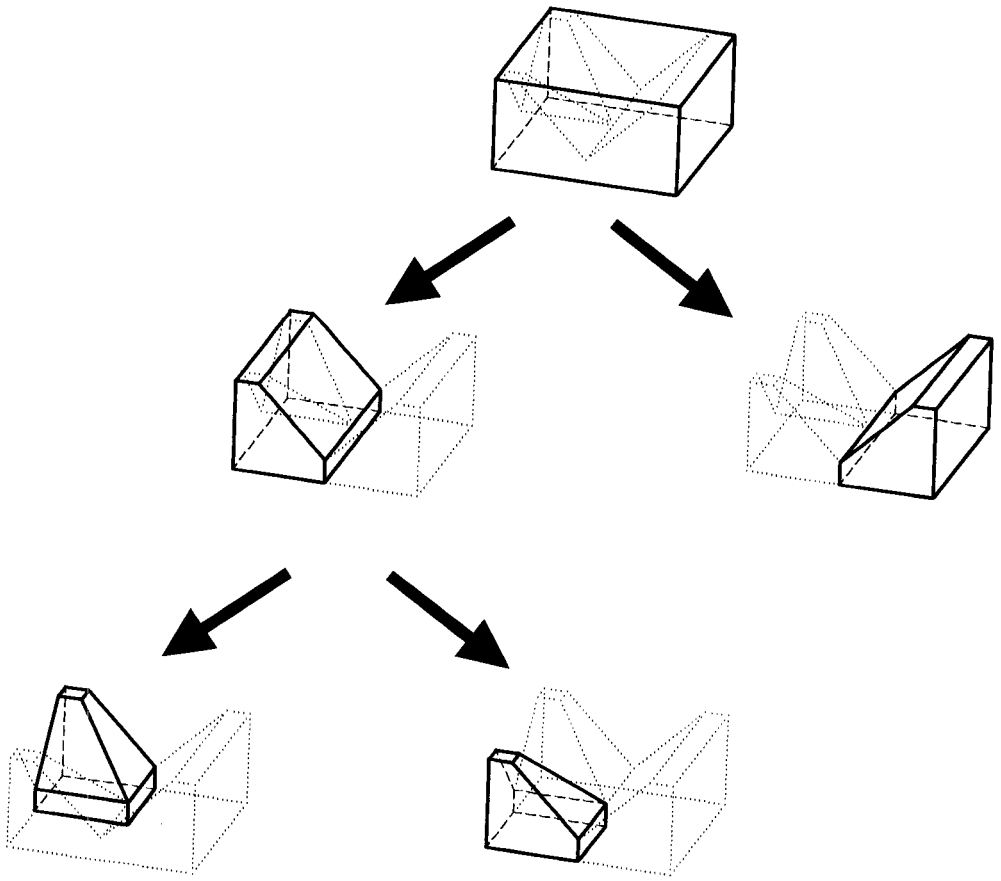


FIGURE 4.2 – Hiérarchie de polyèdres convexes décrivant un volume non convexe (fig. 4.1).

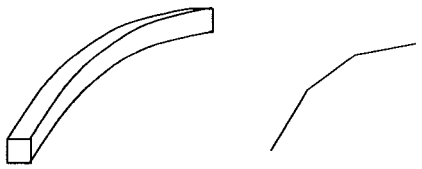


FIGURE 4.3 – Modèle réel et modèle filaire associé

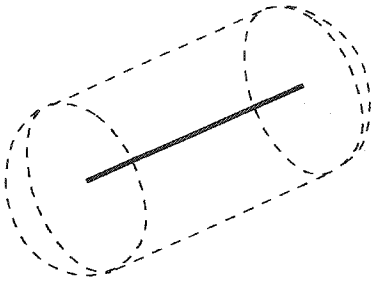


FIGURE 4.4 – Représentation de la matière entourant un segment du modèle filaire

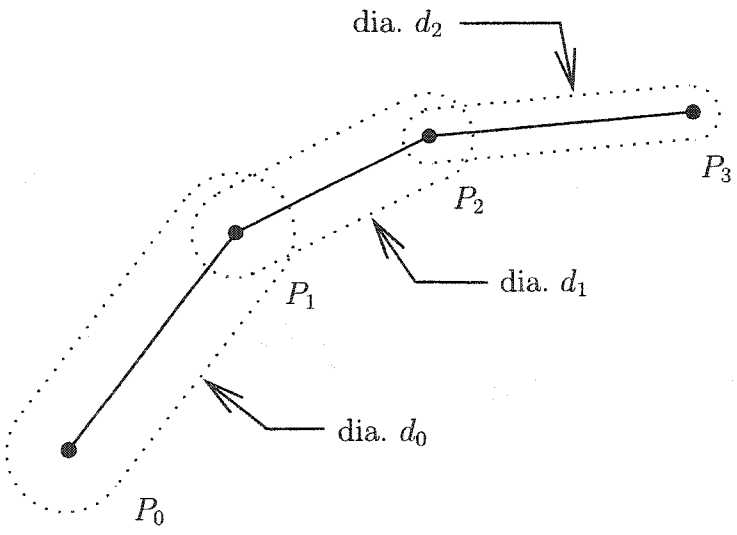


FIGURE 4.5 – Exemple 2D d'un modèle filaire

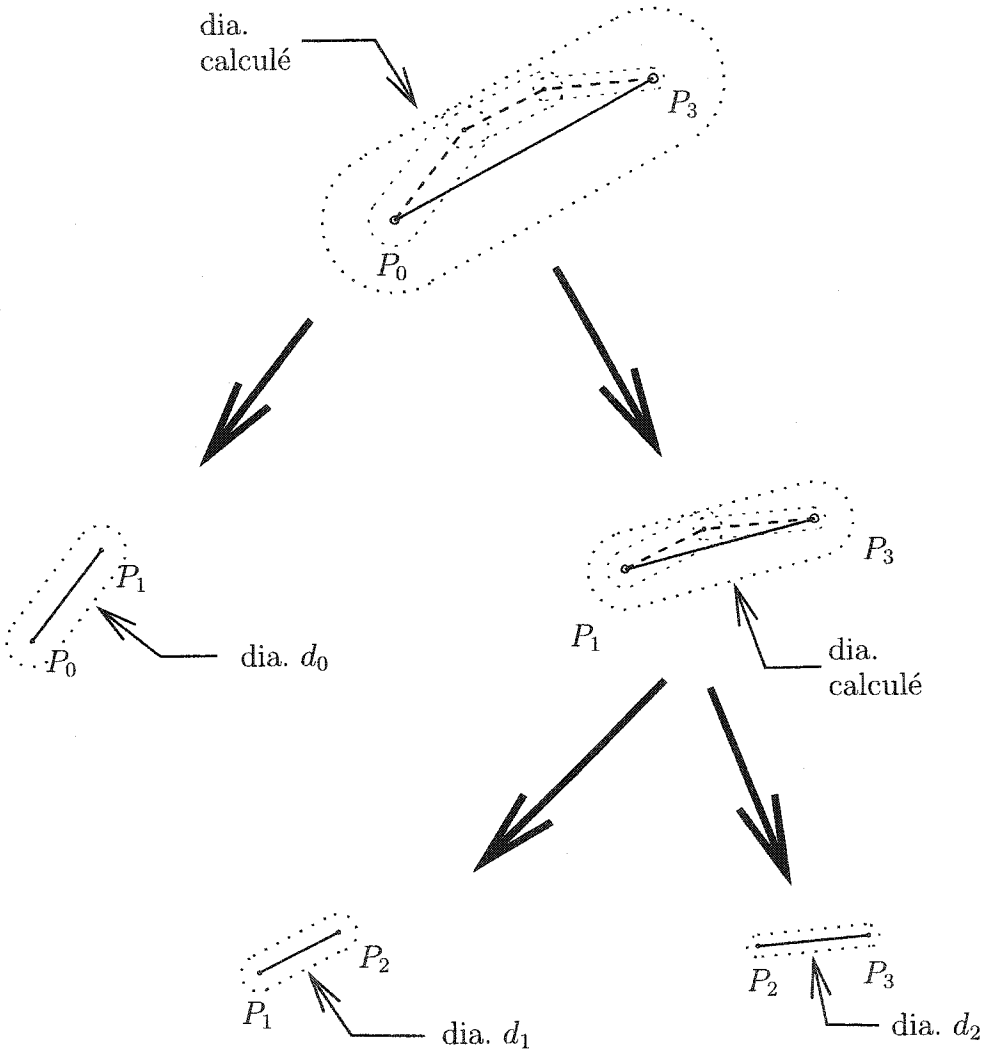


FIGURE 4.6 – Décomposition hiérarchique du modèle de la figure 4.5

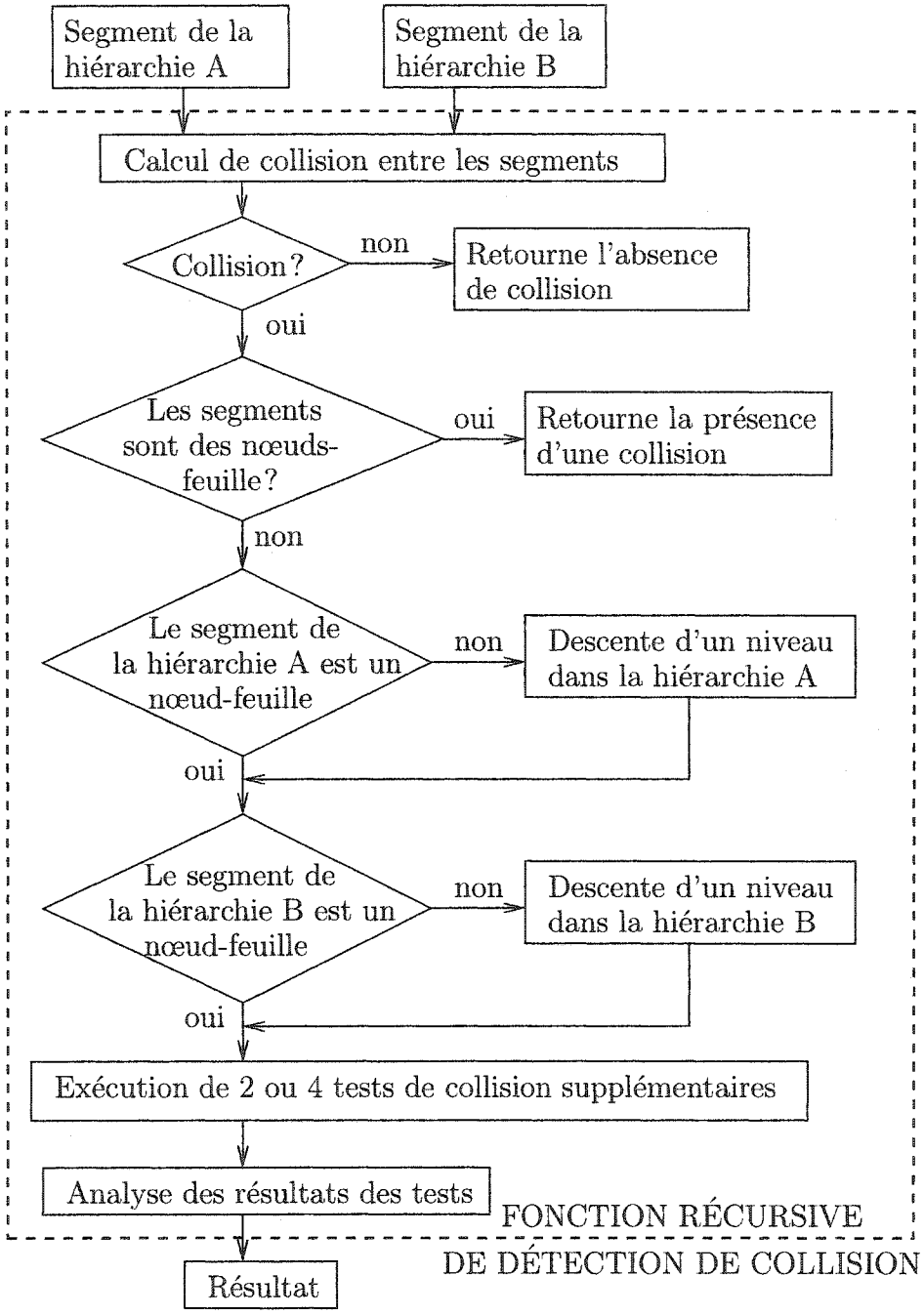


FIGURE 4.7 – Organigramme de fonctionnement de la fonction récursive de détection de collision

CHAPITRE 5

ANALYSE QUALITATIVE DE L'INFLUENCE DES DIFFÉRENTS PARAMÈTRES GÉOMÉTRIQUES SUR L'ESPACE DE TRAVAIL

5.1 But et limites de l'analyse

Le but de ce chapitre est de présenter de manière qualitative l'influence des différents paramètres géométriques sur la forme de l'espace de travail, avec et sans détection de collision. Toutefois, cette analyse qualitative ne peut être que limitée car elle ne peut pas s'étendre à l'ensemble des combinaisons possibles de paramètres.

Afin de faciliter la visualisation et l'interprétation, tous les manipulateurs traités dans les prochaines sections sont symétriques.

Par ailleurs, le code source du simulateur ayant permis la génération des figures présentées dans ce chapitre est disponible sur demande, sous les conditions énoncées dans l'annexe III.

5.2 Influence de la longueur des jambes

La longueur des jambes $(\beta_k + \gamma_k)$ est de loin le paramètre influençant le plus la taille de l'espace de travail du manipulateur. En effet, l'aptitude de chacune des jambes à décrire la sphère sur laquelle se déplacent les joints distaux est déterminante. Plus la longueur de chaque jambe est grande, plus la portion de sphère décrite est grande. Celle-ci est complètement décrite lorsque la jambe a une longueur totale de 180° où chacune des membrures forme un angle de 90°. Dans ce paragraphe, les

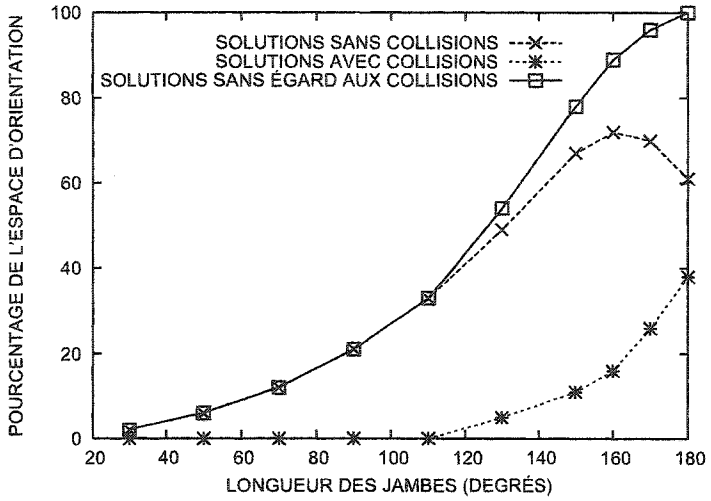


FIGURE 5.1 – Pourcentage de l'espace d'orientation couvert par les solutions (avec collisions, sans collisions et sans égard aux collisions) en fonction de la longueur des jambes $(\beta_k + \gamma_k)$

membrures proximale et distale sont de même dimension, c'est-à-dire que $\beta_k = \gamma_k$.

Le graphique à la figure 5.1 montre le pourcentage de l'espace d'orientation couvert par le manipulateur en fonction de la longueur des jambes $\beta_k + \gamma_k$, c'est-à-dire la proportion des orientations (ψ_1,ψ_2,ψ_3) effectivement atteignables par rapport à l'ensemble des orientations possibles, pour tous les modes d'opération. La taille de l'espace de travail est évaluée de cette manière dans l'ensemble du travail présenté dans ce mémoire. Les trois courbes de la figure 5.1 présentent les solutions menant à des collisions, celles pour lesquelles aucune obstruction n'est constatée et les solutions sans égard aux collisions[1].

Augmenter la dimension des membrures a une influence positive mais seulement jusqu'à un certain point. En effet, plus les membrures sont longues, plus les risques de collision sont élevés. Après une longueur totale de 160° $(\beta_k = \gamma_k = 80°)$, l'augmentation devient néfaste pour la taille de l'espace de travail.

1. c'est-à-dire que la détection de collision n'a pas été utilisée.

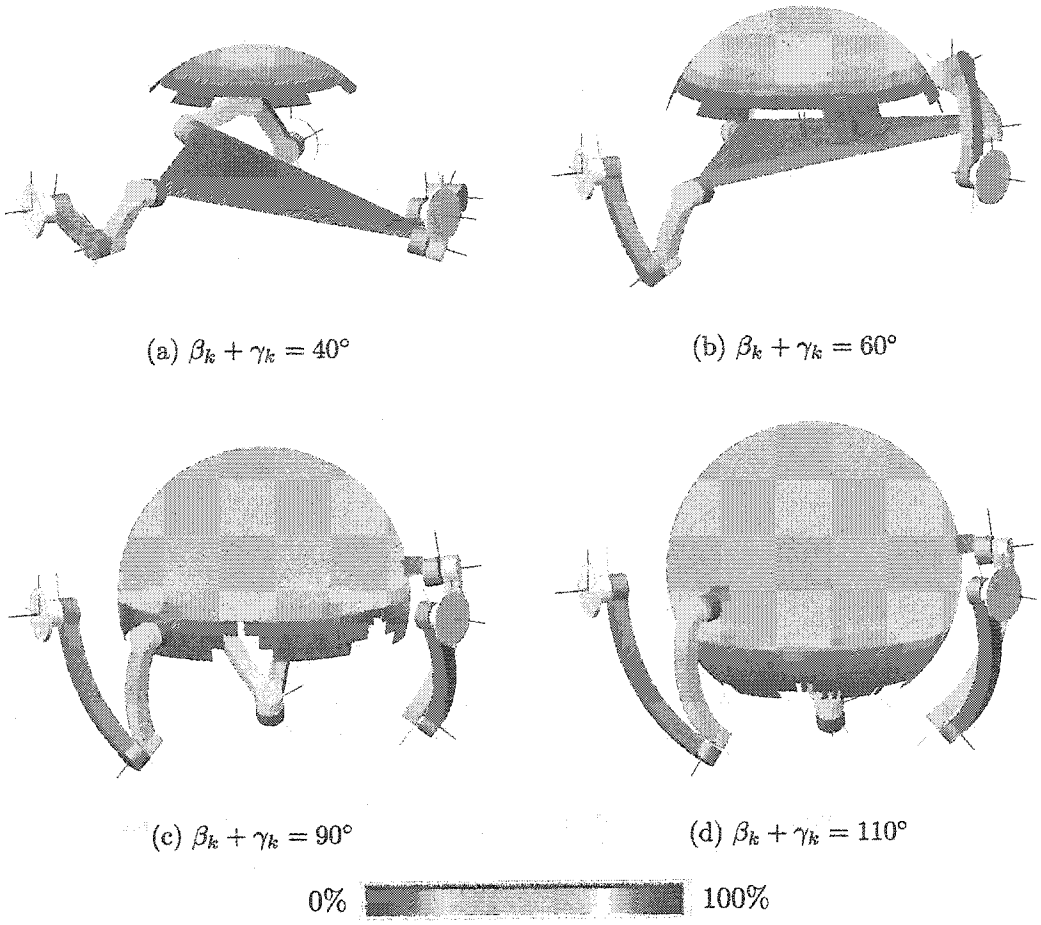


FIGURE 5.2 – Forme de l'espace de travail lorsque les jambes sont trop courtes pour générer des collisions ($\beta_k + \gamma_k < 110°$)

Le manipulateur symétrique étudié a tous les angles du bâti (α_{kl}) et de l'effecteur (δ_{kl}) égaux à 120°. Lorsque les jambes sont plus courtes que 110° ($\beta_k + \gamma_k < 110°$), il n'y a aucune collision (figure 5.2). L'espace de travail a alors tendance à s'étendre de plus en plus, c'est-à-dire que l'angle ψ_2 peut couvrir une plus grande partie de l'intervalle $]-\pi;\pi]$. Par ailleurs, avec des jambes plus grandes, à chaque position (ψ_1,ψ_2), une plus grande partie de l'intervalle $]-\pi;\pi]$ est couverte par l'angle ψ_3.

Au delà de $\beta_k + \gamma_k = 110°$ environ, les jambes deviennent suffisamment grandes

pour générer des collisions. La progression de la forme de l'espace de travail en est alors bien différente tel que montré sur la figure 5.3. La portion de sphère pour $-\frac{\pi}{2} < \psi_2 < \frac{\pi}{2}$ est la plus touchée par l'augmentation du nombre de collisions. En effet, dans cette zone, ψ_3 ne peut couvrir qu'environ 50% de l'intervalle $]-\pi;\pi]$. La ceinture délimitée par $\frac{\pi}{2} < \psi_2 < \pi \cup -\pi < \psi_2 < -\frac{\pi}{2}$ est la seule à profiter de l'allongement des jambes car celles-ci sont beaucoup mieux dépliées que dans la zone $-\frac{\pi}{2} < \psi_2 < \frac{\pi}{2}$. Malheureusement, cette partie de l'espace de travail n'est pas la plus importante pour l'application visée.

5.3 Influence de la position du joint intermédiaire sur les jambes du manipulateur

La position idéale du joint intermédiaire est à mi-chemin, c'est-à-dire que les membrures proximale et distale ont le même angle. En effet, dès que le joint intermédiaire est déplacé, une certaine zone de la sphère décrite par le joint distal n'est pas atteignable par la jambe. En pratique, lorsque le joint est à mi-chemin, un effet néfaste pour le manipulateur peut se produire : les joints distal et proximal peuvent devenir coaxiaux (jambe complètement repliée). Une singularité sérielle est alors atteinte car le moteur rattaché à la jambe peut tourner sans avoir aucun effet sur l'effecteur.

Pour illustrer l'influence du paramètre sur la taille et la forme de l'espace de travail, le manipulateur symétrique ayant tous les angles du bâti (α_{kl}) et de l'effecteur (δ_{kl}) égaux à 120° et les jambes de longueur $\beta_k = \gamma_k = 110°$ est présenté. L'influence de la position du joint intermédiaire sur la taille de l'espace de travail est présentée sur la figure 5.4. Lorsque les membrures proximale et distale ont la même longueur, la taille de l'espace de travail est maximisée. Pour tous les essais réalisés, aucune collision n'est apparue. La réduction de l'espace de travail est donc directement reliée à l'incapacité des joints distaux de se rapprocher des joints proximaux.

L'impact sur la forme de l'espace de travail est moins visible sur l'espace de travail projeté que sur la représentation en trois dimensions. La figure 5.5 présente l'évolution de la forme en partant d'un ratio de 1 vers un ratio de 4. La forme extérieure reste constante mais trois trous cylindriques apparaissent dans la direction de l'axe ψ_2. Par ailleurs, dans la direction $\psi_1 + \psi_3 = 0$, un autre trou apparaît. Ceux-ci correspondent aux positions pour lesquelles les joints distaux ont tendance à se rapprocher des joints proximaux, ce qui n'est plus permis à cause des membrures proximales de longueur différente des membrures distales.

5.4 Influence de la forme du bâti

Le graphique à la figure 5.6 montre l'impact de la forme du bâti, c'est-à-dire de la valeur des angles α_{kl} sur la taille de l'espace de travail. Le manipulateur utilisé a des jambes de dimensions $\beta_k = \gamma_k = 65°$ et un effecteur ayant tous ses angles $\delta_{kl} = 120°$.

À mesure que les joints proximaux s'éloignent, c'est-à-dire lorsque les angles α_{kl} grandissent, le risque de collision entre les membrures proximales diminue. La taille de l'espace de travail, c'est-à-dire la quantité de solutions sans collisions, a donc tendance à augmenter sensiblement. Par contre, si aucune technique de détection de collision n'était utilisée, avoir des membrures plus rapprochées serait bénéfique sur le nombre de solutions du modèle géométrique inverse. Ces deux résultats contradictoires prouvent encore l'intérêt d'utiliser la détection de collisions dans le calcul d'espaces de travail.

Il devrait être possible d'obtenir des manipulateurs avec des jambes plus longues et des joints proximaux très écartés ayant la même dimension d'espace de travail (mais pas la même forme) que des manipulateurs avec des jambes courtes et des

joints proximaux rapprochés.

Répéter les mêmes essais sur un manipulateur avec des jambes plus courtes ($\beta_k = \gamma_k = 45°$) donne un tout autre résultat (figure 5.7). Pour toute la gamme d'angle présentée, il n'y a aucune collision sauf pour $\alpha_{kl} = 45°$ mais leur nombre est négligeable. Contrairement à l'exemple précédent, le nombre de solutions augmente à mesure que la distance entre les joints proximaux croît. Il n'est donc pas possible de connaître clairement l'influence de la forme du bâti dans la dimension de l'espace de travail en faisant totalement abstraction des autres paramètres géométriques.

L'influence des α_{kl} sur la forme d'espace de travail est présentée sur la figure 5.8. Rapprocher les joints proximaux a tendance à étaler un peu plus les solutions sur la sphère décrite par ψ_1 et ψ_2. En effet, moins de solutions sont disponibles autour de valeurs faibles de $\psi 2$ mais d'autres positions avec des valeurs élevées de ψ_2 deviennent atteignables.

5.5 Influence de la forme de l'effecteur

D'un point de vue exclusivement cinématique, c'est-à-dire sans tenir compte des collisions, l'influence de la forme de l'effecteur est tout à fait identique à celle du bâti. En effet, il suffit d'intervertir le rôle du bâti et de l'effecteur pour le constater. La figure 5.9 présente l'impact de la valeur des δ_{kl} pour un manipulateur avec des jambes de dimensions $\beta_k = \gamma_k = 65°$ et un bâti ayant tous ses angles égaux à 120°. Elle est pratiquement superposable à la figure 5.6 de la section précédente. La légère différence, visible pour $\delta_{kl} = 115°$, provient du nombre de collisions différent dans les deux cas de figure qui pourrait provenir des imprécisions du modèle filaire utilisé et des calculs numériques.

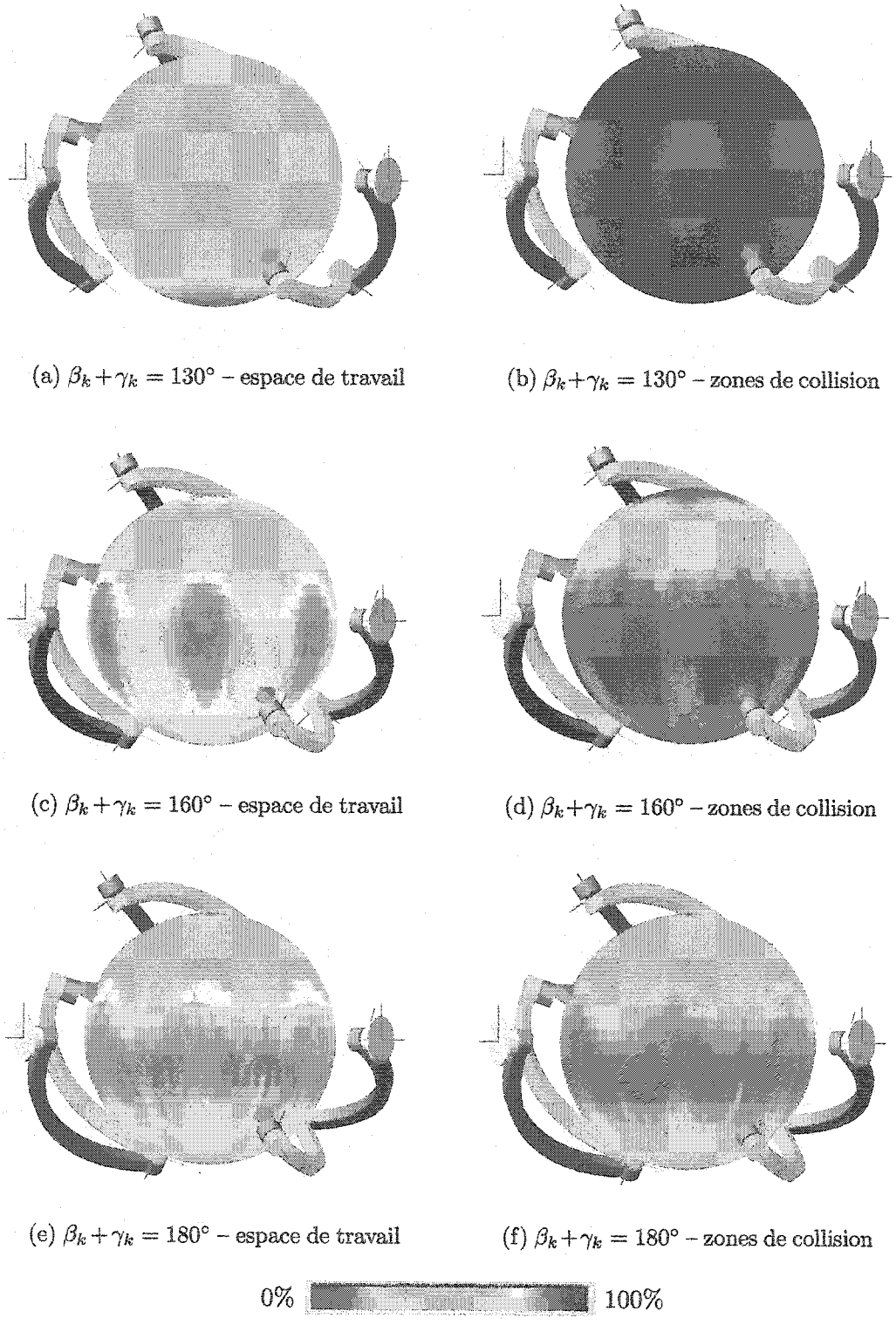
Un deuxième essai similaire à celui réalisé dans la section précédente montre des

résultats très semblables. La figure 5.10 est quasiment superposable à la figure 5.7. Le manipulateur utilisé avait cette fois-ci des jambes de dimensions $\beta_k = \gamma_k = 45°$.

Si la dimension de l'espace de travail est quasi-identique, il n'en est pas de même pour sa forme. La figure 5.11 présente la forme de l'espace de travail en 3D et projeté pour les même valeurs que la figure 5.8 mais appliquées aux angles δ_{kl} plutôt que α_{kl}. La forme de l'espace de travail est beaucoup plus creusée avec des zones inatteignables pour de petites valeurs de ψ_2. Ceci a un impact vraiment négatif sur l'utilisation du manipulateur dans l'application visée.

5.6 Conclusion de l'analyse

L'analyse manuelle de l'influence de douze paramètres indépendants n'est en fait pas possible du fait de la complexité des équations mises en jeu. Ainsi, même si elle a permis d'extraire de grandes tendances dans le comportement des paramètres, elle est insuffisante pour permettre de choisir convenablement la valeur de chacun des paramètres pour une application donnée. Il faut donc faire appel à un processus automatisé d'optimisation.



(a) $\beta_k + \gamma_k = 130°$ – espace de travail

(b) $\beta_k + \gamma_k = 130°$ – zones de collision

(c) $\beta_k + \gamma_k = 160°$ – espace de travail

(d) $\beta_k + \gamma_k = 160°$ – zones de collision

(e) $\beta_k + \gamma_k = 180°$ – espace de travail

(f) $\beta_k + \gamma_k = 180°$ – zones de collision

FIGURE 5.3 – Appartion de collisions lorsque la longueur des jambes $\beta_k + \gamma_k$ augmente

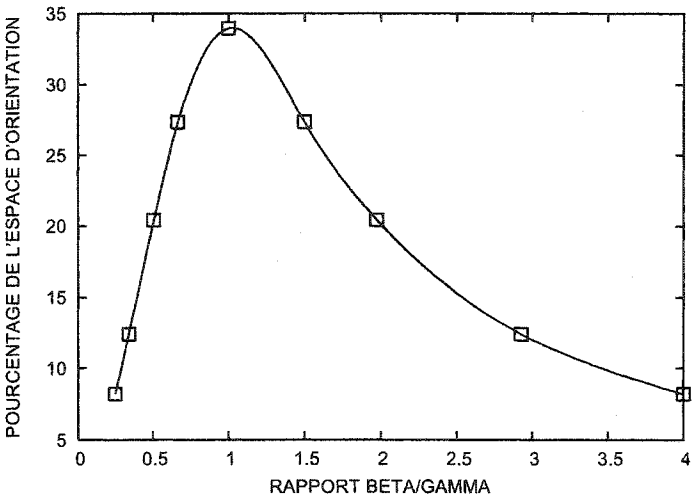
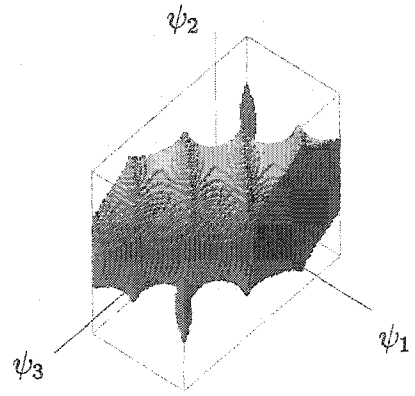
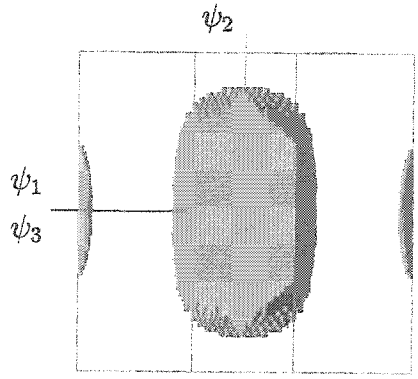


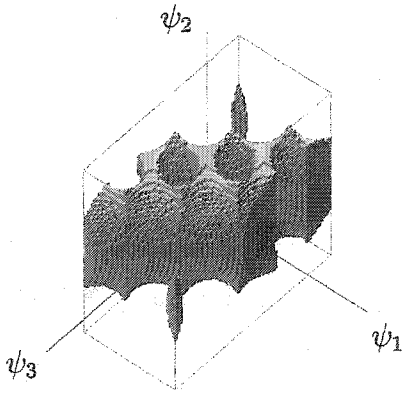
FIGURE 5.4 – Pourcentage de l'espace d'orientation couvert par les solutions sans collisions en fonction du ratio β_k/γ_k



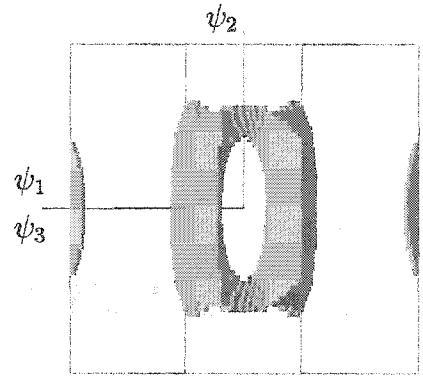
(a) $\beta_k/\gamma_k = 55°/55°$ – vue 1



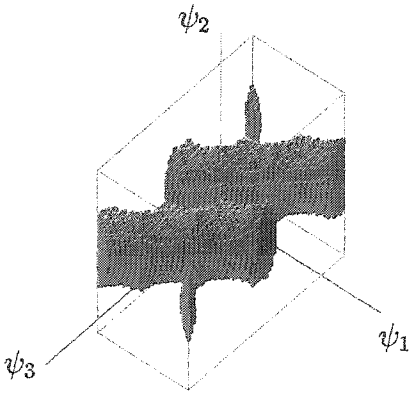
(b) $\beta_k/\gamma_k = 55°/55°$ – vue 2



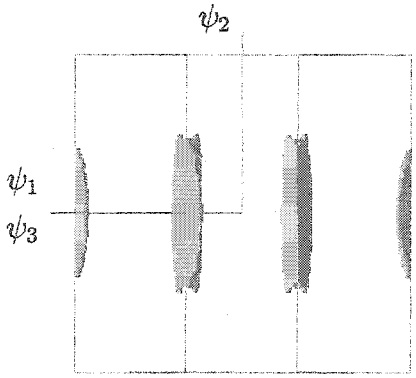
(c) $\beta_k/\gamma_k = 73°/37°$ – vue 1



(d) $\beta_k/\gamma_k = 73°/37°$ – vue 2



(e) $\beta_k/\gamma_k = 88°/22°$ – vue 1



(f) $\beta_k/\gamma_k = 88°/22°$ – vue 2

FIGURE 5.5 – Influence du ratio β_k/γ_k sur la forme de l'espace de travail

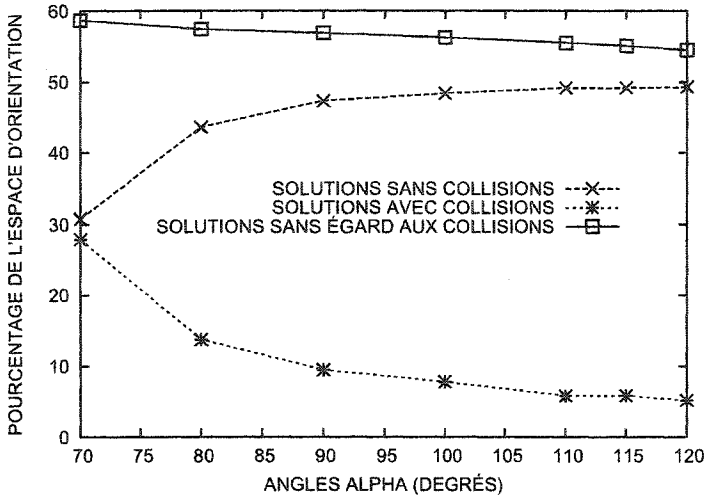


FIGURE 5.6 – Pourcentage de l'espace d'orientation couvert par les solutions (avec collisions, sans collisions et sans égard aux collisions) en fonction de la forme du bâti (angles α_{kl}) avec $\beta_k = \gamma_k = 65°$

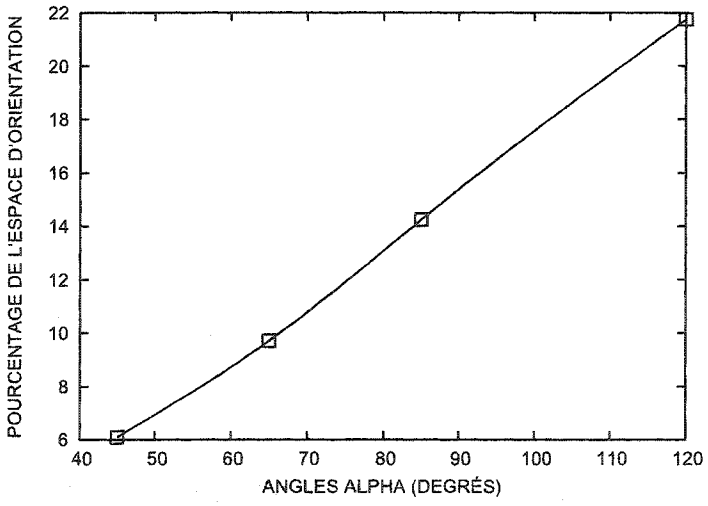
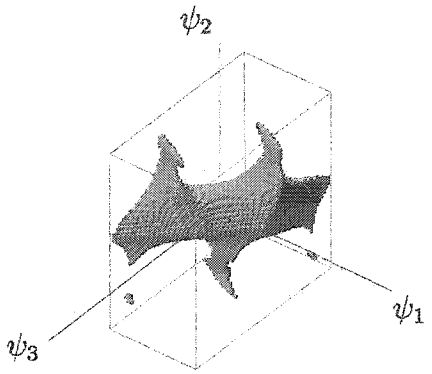
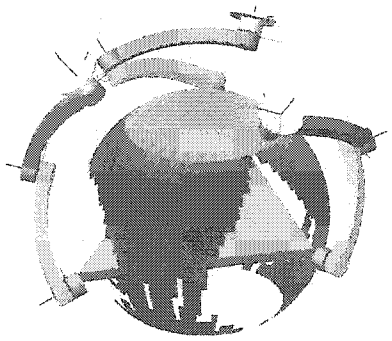


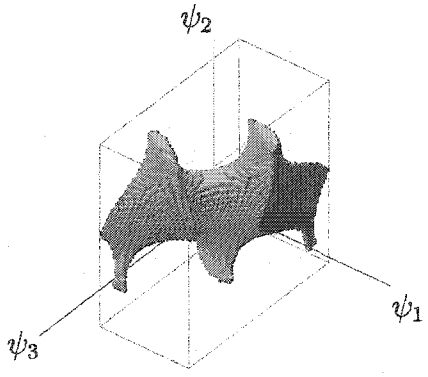
FIGURE 5.7 – Pourcentage de l'espace d'orientation couvert par les solutions sans collisions en fonction de la forme du bâti (angles α_{kl}) avec $\beta_k = \gamma_k = 45°$



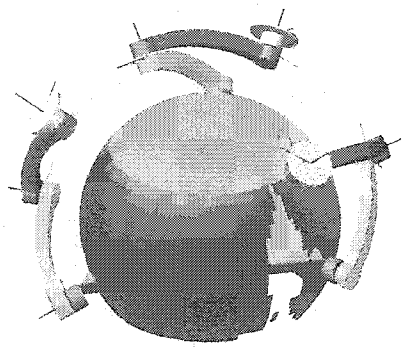
(a) $\alpha_{kl} = 65°$ – espace de travail



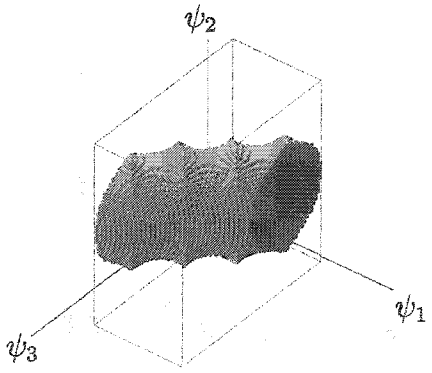
(b) $\alpha_{kl} = 65°$ – espace projeté



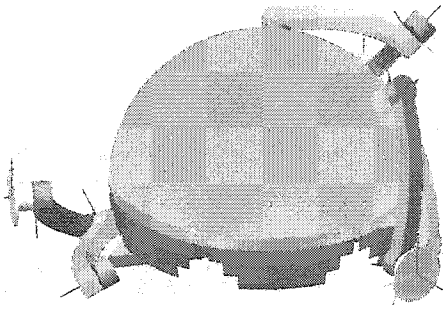
(c) $\alpha_{kl} = 85°$ – espace de travail



(d) $\alpha_{kl} = 85°$ – espace projeté



(e) $\alpha_{kl} = 120°$ – espace de travail



(f) $\alpha_{kl} = 120°$ – espace projeté

FIGURE 5.8 – Influence des angles α_{kl} sur la forme de l'espace de travail

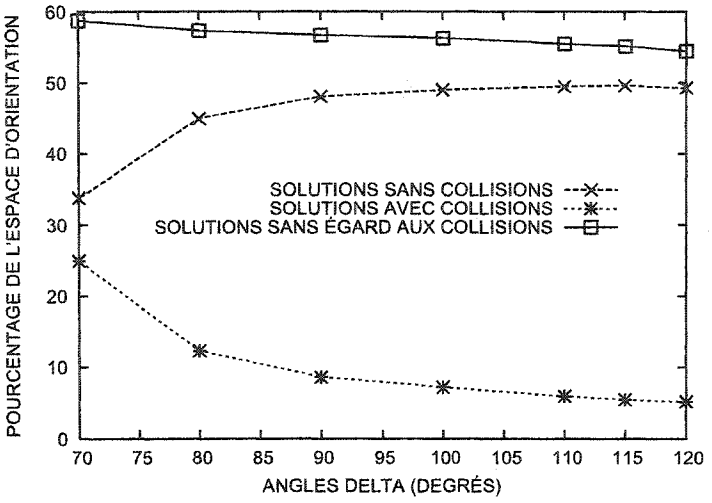


FIGURE 5.9 – Pourcentage de l'espace d'orientation couvert par les solutions (avec collisions, sans collisions et sans égard aux collisions) en fonction de la forme de l'effecteur (angles δ_{kl}) avec $\beta_k = \gamma_k = 65°$

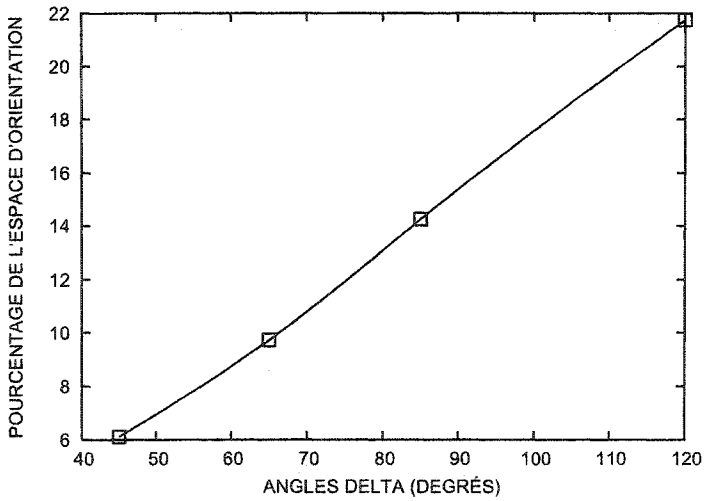
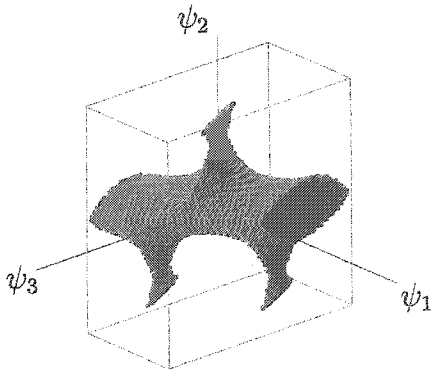
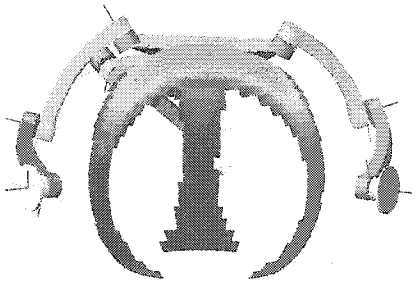


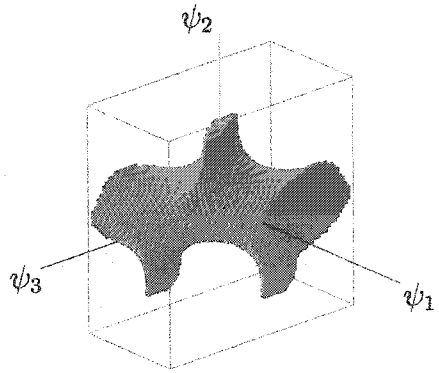
FIGURE 5.10 – Pourcentage de l'espace d'orientation couvert par les solutions sans collisions en fonction de la forme de l'effecteur (angles δ_{kl}) avec $\beta_k = \gamma_k = 45°$



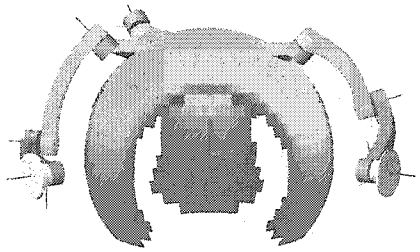
(a) $\delta_{kl} = 65°$ – espace de travail



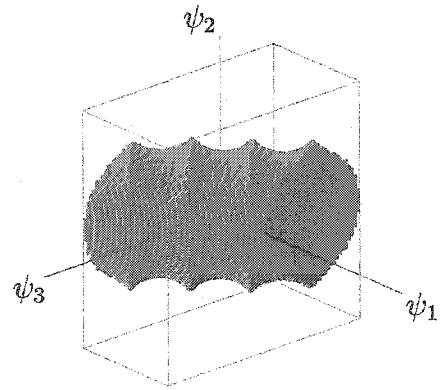
(b) $\delta_{kl} = 65°$ – espace projeté



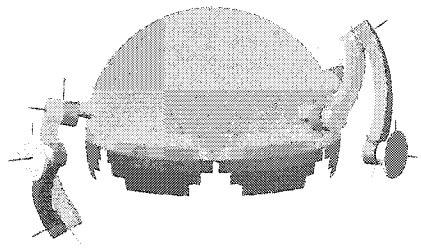
(c) $\delta_{kl} = 85°$ – espace de travail



(d) $\delta_{kl} = 85°$ – espace projeté



(e) $\delta_{kl} = 120°$ – espace de travail



(f) $\delta_{kl} = 120°$ – espace projeté

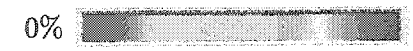


FIGURE 5.11 – Influence des angles δ_{kl} sur la forme de l'espace de travail

CHAPITRE 6

SYNTHÈSE GÉOMÉTRIQUE À L'AIDE D'ALGORITHMES GÉNÉTIQUES CODÉS RÉELS

6.1 Choix de la technique d'optimisation et justification

Le problème d'optimisation consiste à trouver une ou plusieurs combinaisons optimales de paramètres géométriques pour l'application visée. Les méthodes déterministes telles que NEWTON-RAPHSON ne peuvent pas être aisément utilisées car l'optimisation porte sur neuf paramètres indépendants. D'un autre côté, les méthodes stochastiques telles que Monte-Carlo sont d'un intérêt relativement limité car elles laissent trop de place au hasard. Le domaine de recherche est grand d'où une probabilité relativement faible de trouver une solutions optimale rapidement.

La méthode des algorithmes génétiques est semi-déterministe. Bien que laissant une bonne part au hasard, elle a l'avantage de conserver les meilleures solutions de l'itération précédente afin de les comparer avec les nouvelles solutions trouvées. Ainsi, la convergence peut être rapide. L'aspect aléatoire permet en outre d'explorer d'autres portions du domaine de recherche. Le champ d'utilisation des algorithmes génétiques est plutôt large. Ils deviennent une méthode de choix pour les espaces de recherches de grande dimension, discontinus, complexes et peu connus. Même s'ils ne peuvent pas garantir l'obtention de la solution optimale globale, ils sont généralement bons pour obtenir des solutions de qualité raisonnable rapidement.

Les algorithmes génétiques ont déjà été utilisés dans le domaine de la robotique pour la synthèse géométrique donnant de bons résultats (TREMBLAY, 1999; BARON,

2001; BERNIER, 2003).

Le code source du programme d'optimisation est disponible sur demande, sous les conditions énoncées dans l'annexe III.

6.2 Présentation générale des algorithmes génétiques et revue bibliographique

Les algorithmes génétiques se basent sur le modèle de sélection naturelle en biologie. La technique consiste à avoir une population de chromosomes qui représentent les solutions à un problème. Chaque chromosome représente un individu qui, dans l'application visée est l'ensemble des neuf paramètres géométriques décrivant un manipulateur. À chaque individu est associé un critère de performance calculé par une fonction objectif adaptée aux performances recherchées.

Afin d'assurer une convergence vers des solutions meilleures, en espérant arriver à des solutions optimales, les algorithmes génétiques partent du postulat que de bons individus peuvent donner de bons enfants, voire meilleurs. Une explication mathématique est disponible dans le livre de GOLDBERG (1989). Ainsi, à chaque génération, des individus sont choisis au hasard mais les individus les meilleurs ont une plus grande probabilité d'être sélectionnés. Les deux individus sont ensuite croisés pour donner deux enfants. Le croisement (fig. 6.2) associé au critère de performance donne l'aspect déterministe aux algorithmes génétiques. En effet, les nouveaux individus et les anciens sont ensuite classés par ordre décroissant de performance et les individus les plus faibles sont éliminés. À chaque génération, la population devient donc toujours plus performante. La figure 6.1 représente les grandes étapes pour une génération.

Un opérateur de mutation (fig. 6.3) calqué sur ce qui se produit couramment dans

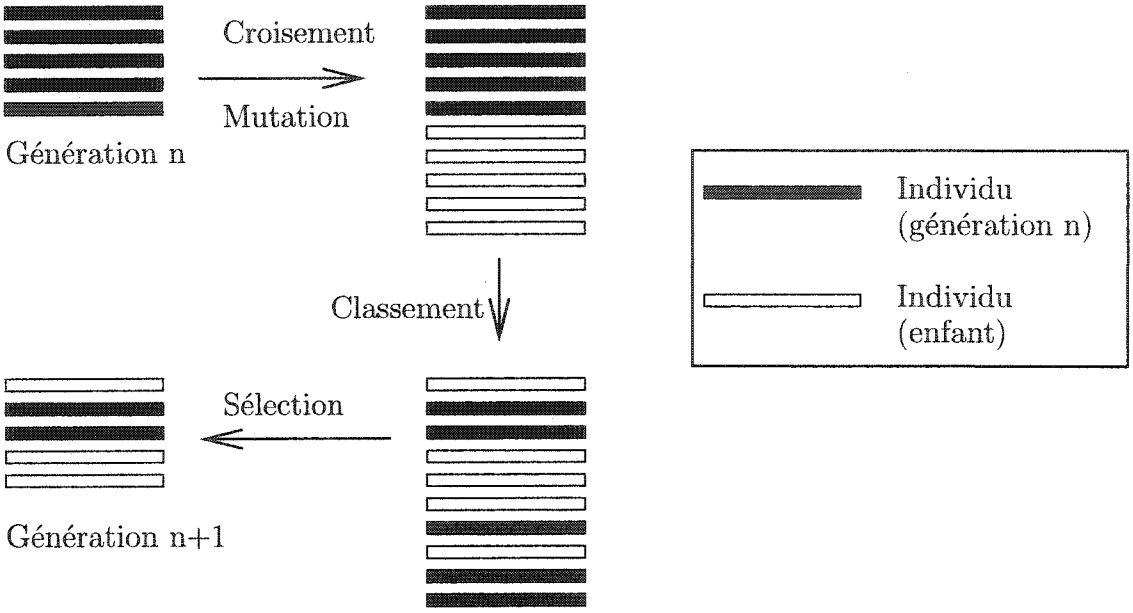


FIGURE 6.1 – Une génération dans un algorithme génétique.

la nature est aussi programmé. Celui-ci permet d'explorer des solutions du domaine de recherche de manière purement aléatoire. Celui-ci est généralement appliqué au hasard sur une faible portion de la population.

À l'origine, les algorithmes génétiques étaient directement calqués sur ce qui se produit dans la nature. Lorsque deux chromosomes sont sélectionnés pour le croisement, un lieu de croisement, le *locus*, est déterminé au hasard sur les chromosomes. Le matériel génétique est échangé par les deux chromosomes et donne donc deux chromosomes avec un ensemble de gènes uniques. Comme le *locus* peut se trouver n'importe où sur le chromosome, soit entre deux gènes ou bien en plein milieu d'un gène, les premières implémentations des algorithmes génétiques ont été réalisées de manière à reproduire à l'identique ce phénomène. Ceci a mené à l'utilisation des algorithmes codés binaires où chaque chromosome est une chaîne de bits dans laquelle des gènes sont positionnés. À chaque gène correspond un nombre de bits constant. En choisissant un lieu de croisement au hasard dans la chaîne de bits, le croisement naturel est reproduit.

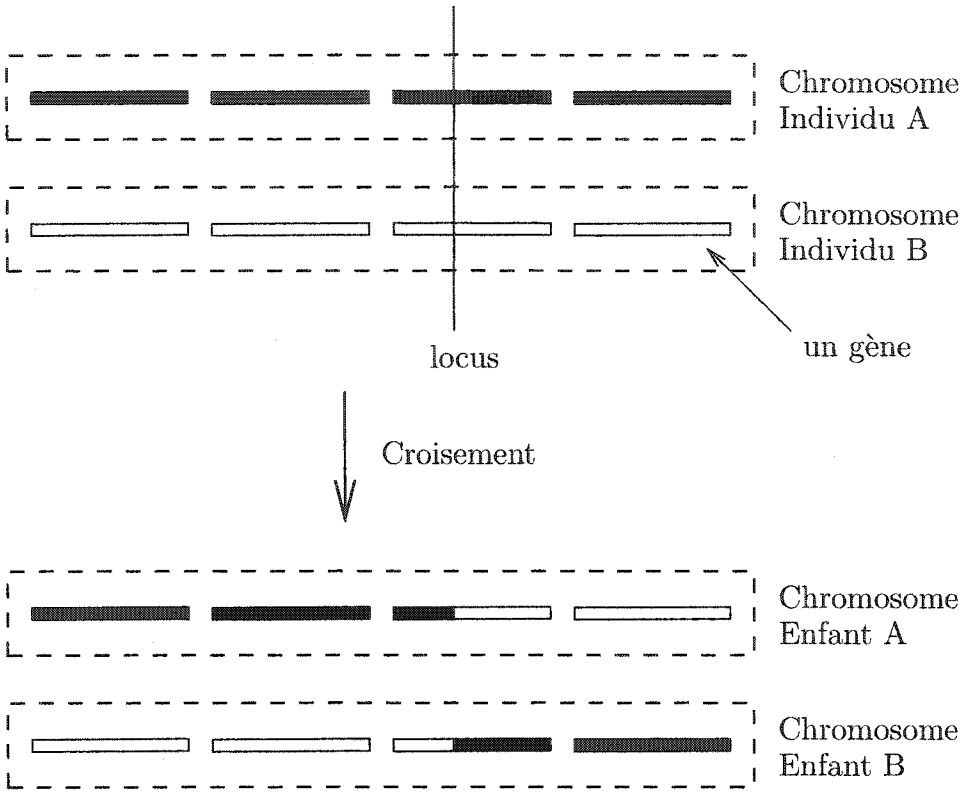


FIGURE 6.2 – Représentation du croisement.

Toutefois, deux inconvénients majeurs apparaissent avec les algorithmes génétiques codés binaires. D'une part, les gènes n'étant définis que par un nombre limité de bits, il faut choisir l'intervalle dans lequel le paramètre associée va se confiner. D'autre part, pour un même nombre de bits, seulement un nombre fini de valeurs peut être atteint. Plus l'intervalle est grand, plus la distance entre deux valeurs contiguës est grande. Ainsi, il peut se produire plusieurs cas de figure :

- la solution optimale recherchée se trouve en dehors de l'intervalle sélectionné pour le gène concerné;
- la solution optimale se trouve à une valeur intermédiaire non codée par l'ensemble de bits associé au gène.

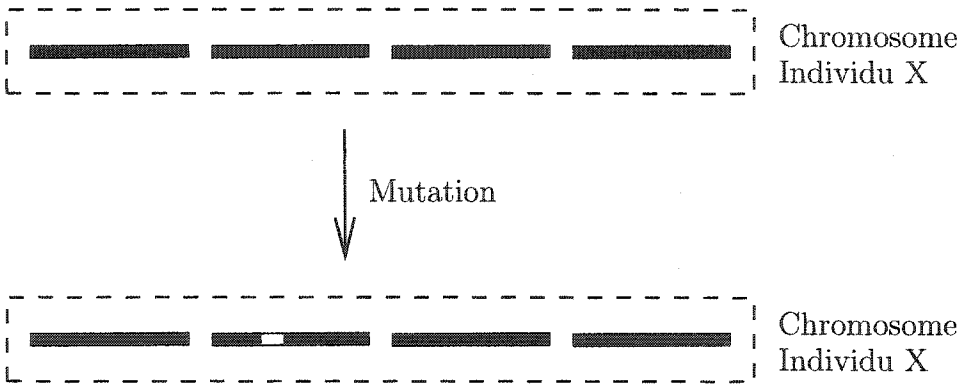


FIGURE 6.3 – Représentation de la mutation.

Pour contrer ces phénomènes, il faut agrandir le domaine de recherche tout en raffinant la discrétisation de l'intervalle de recherche en ajoutant des bits supplémentaires au gène concerné.

Ainsi, pour éliminer ces défauts majeurs liés aux algorithmes génétiques codés binaires, une autre version a été développée, plus éloignée de l'origine biologique dans la représentation des gènes mais plus efficace. Elle fait appel à un codage en nombre réel à virgule flottante qui permet non seulement de s'affranchir du choix d'un intervalle de recherche pour chaque bits mais en plus permet d'obtenir une plus grande précision. Il est donc possible de laisser évoluer les individus dans un domaine de recherche totalement inconnu sans prendre le risque d'éliminer la zone où se trouve la ou les solutions optimales. Chaque gène est représenté par un nombre réel à virgule flottante. Un individu est donc représenté par un tableau de nombres réels. De nombreux opérateurs simulant le croisement et la mutation ont été développés pour cette nouvelle représentation.

L'article de HERRERA, LOZANO et VERDEGAY (1998) est un excellent point de départ pour les deux types d'algorithmes génétiques expliqués car il présente un grand éventail d'opérateur et une analyse détaillée des avantages et des problèmes

rencontrés avec les algorithmes génétiques.

Il faut noter que les explications fournies dans ce paragraphe sont très générales car il existe une multitude de variantes qui ont été développées et testées pour répondre à des problèmes spécifiques. L'un des problèmes majeurs rencontrés avec les algorithmes génétiques est la convergence prématurée vers une solution qui n'est qu'un optimum local. Toutefois, en utilisant plusieurs critères de performance et des opérateurs de croisement et de mutation adaptés, il est possible de réduire ce phénomène.

6.3 Choix des opérateurs de croisement et de mutation

Parmi les nombreux opérateurs de croisement plusieurs ont été essayés :

- Le croisement simple où un lieu de croisement est choisi dans le tableau représentant le chromosome des individus. Le croisement correspond alors à la même chose que la figure 6.2 à la seule différence que le locus ne peut pas se trouver au milieu d'un gène. Les gènes des enfants ont donc des valeurs identiques à l'un des deux parents.
- Le croisement BLX-α où la valeur de chaque gène est choisie aléatoirement et uniformément dans un intervalle $[c_{min} - I\,\alpha; c_{max} + I\,\alpha]$. Les valeurs c_{min} et c_{max} correspondent respectivement à la valeur minimale et à la valeur maximale du gène pour les parents sélectionnés et I représente la distance $c_{max} - c_{min}$. La valeur de α est à choisir en fonction du comportement recherché.

Le croisement simple ne donnait pas de résultat intéressant et menait à une convergence précoce vers des solutions non optimales. En effet, comme chaque enfant

récupère exactement, pour chaque gène, la valeur de l'un des deux parents, il n'y a pas de nouvelles valeurs créées, à moins qu'il y ait eu une mutation. L'intérêt d'utiliser des algorithmes génétiques codés réels est alors complètement perdu.

Par contre, le croisement BLX-α est d'une grande efficacité malgré sa simplicité. En choisissant une valeur positive pour α, on permet une exploration plus importante du domaine de recherche. Toutefois, il ne faut pas trop agrandir l'intervalle car la convergence s'en trouve réduite assez rapidement. Une valeur $\alpha = 0.5$ est généralement un bon compromis entre l'exploration du domaine de recherche et une bonne convergence du calcul.

La probabilité d'une mutation ne doit pas être trop élevée car l'intérêt des algorithmes génétiques serait alors perdu et une méthode totalement stochastique serait plus appropriée. Ce facteur a été ajusté à 6% et 7% lors des calculs. Toutefois, afin de ne pas perdre les meilleurs individus, ceux-ci sont systématiquement épargnés de toute mutation.

Une mutation purement aléatoire a d'abord été essayée. Celle-ci consiste à prendre un gène au hasard dans le chromosome de l'individu choisi puis de lui donner une valeur aléatoire prise dans l'intervalle pour lequel il sera valide. Malheureusement, l'individu mutant devenait la plupart du temps invalide car l'ensemble des paramètres géométriques du manipulateur sont liés par les contraintes citées à la fin du chapitre sur le paramétrage du manipulateur.

Un opérateur de mutation réduisant les risques de rendre invalide l'ensemble des paramètres est donc plus approprié. Celui-ci est basé sur des changements de la géométrie du manipulateur :

- changement de la longueur d'une jambe choisie au hasard;
- changement de la position d'un joint intermédiaire sur une jambe;

- changement de la position d'un joint proximal par rapport aux deux autres (déformation du bâti du manipulateur);
- déformation du bâti en conservant les mêmes proportions pour les distances entre les joints proximaux;
- changement de la position d'un joint distal par rapport aux deux autres (déformation de l'effecteur);
- déformation de l'effecteur en conservant les mêmes proportions pour les distances entre les joints distaux;
- rendre le bâti symétrique;
- rendre l'effecteur symétrique;
- rendre toutes les jambes identiques;
- permutation des angles du bâti;
- permutation des angles de l'effecteur;
- permutation des angles des jambes.

Toutes ces modifications sont faites à plusieurs paramètres géométriques de l'individu. Le choix de l'opération à effectuer est aussi aléatoire. Les premières opérations ont généralement tendance à détruire la symétrie du manipulateur d'où l'existence d'opérations pour rendre symétrique un manipulateur dissymétrique. La probabilité de réalisation d'un des deux types d'opération est identique afin de ne pas favoriser l'une ou l'autre possibilité.

6.4 Particularités et remarques supplémentaires

Du fait qu'un ensemble de paramètres peut être invalide du point de vue de la géométrie du manipulateur, la validité de l'ensemble de paramètres est systématiquement vérifiée avant d'ajouter un individu (soit lors de la génération de la po-

pulation, soit lors d'un croisement ou d'une mutation). Tant que l'individu obtenu n'est pas valide, l'opération (génération, croisement ou mutation) est réitérée.

Lorsqu'un ensemble de paramètres est valide, cela ne signifie pas que le manipulateur sera utilisable. En effet, lors de la génération des individus, il arrive fréquemment que les jambes soient trop courtes pour que le manipulateur puisse même s'assembler. Avant d'insérer l'individu dans la population, l'espace de travail est calculé pour savoir s'il n'est pas vide.

La population initiale est générée de manière purement aléatoire, en prenant soin de vérifier les deux points énoncés dans les paragraphes précédents. Par ailleurs, un ensemble de manipulateurs déjà connus pour leur performance acceptable est aussi ajouté.

La sélection des parents lors d'un croisement se fait de manière aléatoire mais les individus les plus performants ont plus de chance d'être choisis. La valeur du critère de performance associée à chaque individu est donc utilisée à cette fin.

6.5 Optimisation à simple critère

La première optimisation réalisée doit permettre de répondre à la question suivante : le manipulateur optimal en terme de volume maximum de l'espace de travail est-il nécessairement symétrique?

Au bout de 695 générations, le calcul portant sur une population de 150 individus a donné le résultat brut suivant :

- $\alpha_{AB} = 117.117°$, $\alpha_{AC} = 118.925°$, $\alpha_{BC} = 121.137°$;
- $\beta_A = 82.841°$, $\gamma_A = 82.556°$;

- $\beta_B = 85.703°$, $\gamma_B = 85.302°$;
- $\beta_C = 83.516°$, $\gamma_C = 84.535°$;
- $\delta_{AB} = 114.377°$, $\delta_{AC} = 117.834°$, $\delta_{BC} = 127.787°$;

La précision des résultats obtenus n'est sûrement pas de l'ordre du millième de degrés. En effet, il faut prendre en compte plusieurs facteurs qui contribuent à l'imprécision des résultats. Premièrement, les algorithmes génétiques ont tendance à converger beaucoup plus lentement vers la fin. Deuxièmement, afin de ne pas allonger le calcul déjà interminable (7 jours sur un serveur de calcul Power4 IBM), la profondeur des *octrees* n'était que de 5, ce qui limite la précision du calcul des espaces de travail. Enfin, les erreurs liées aux calculs numériques et à l'imprécision du modèle filaire utilisé dans la détection de collision sont aussi à prendre en considération.

Ainsi, il est possible de formuler certaines conclusions. La solution optimale trouvée est presque symétrique. Il se pourrait très bien que la dissymétrie proviennent justement des incertitudes du calcul. Ensuite, les jambes ont presque la même longueur et le joint intermédiaire est positionné de manière à ce que les membrures proximales et distales aient la même longueur. Ainsi, le manipulateur optimal aurait des paramètres géométriques peu différents des suivants :

- $\alpha_{kl} = 119°$;
- $\beta_k = \gamma_k = 84°$;
- $\delta_{kl} = 120°$;

Par ailleurs, l'espace de travail du manipulateur occupe environ 76% à 77% de l'espace d'orientation possible.

6.6 Optimisation multicritères pour l'application envisagée — premier essai

Le manipulateur obtenu avec l'optimisation à simple critère de la section précédente n'est pas nécessairement celui qui sera optimal pour l'utilisation dans la machine à commande numérique. Plusieurs modifications sont donc apportées dont l'usage de multiples critères de performance.

6.6.1 Choix des bornes du volume discrétisé

Étant donné que l'effecteur ne pivotera pas à plus de 90° (c'est-à-dire $|\psi_2| \leq 90°$), les bornes du volume discrétisé en vue du calcul de l'espace de travail peuvent être ajustées :

$$\psi_1 \in]-\frac{\pi}{2};\frac{\pi}{2}] \tag{6.1a}$$

$$\psi_2 \in]-\frac{\pi}{2};\frac{\pi}{2}] \tag{6.1b}$$

$$\psi_3 \in]-\pi;\pi] \tag{6.1c}$$

Ainsi, une plus grande précision que le calcul précédent sera obtenue sans avoir à augmenter la profondeur de l'*octree*.

6.6.2 Choix des critères d'optimisation

Les critères d'optimisation sont des grandeurs adimensionnelles, dont le choix et le calcul sont justifiés et expliqués dans la suite de cette section. Ils sont au nombre de 7 :

- Le volume de l'espace de travail;

- Un facteur de forme de l'espace de travail;
- Le volume de l'espace de travail tenant compte des solutions où une ou plusieurs jambes peuvent gêner le parcours de la broche;
- Le degré de symétrie du manipulateur;
- La longueur des jambes;
- La dimension du bâti;
- La dimension de l'effecteur.

Une lettre est associée à chaque critère en référence aux tableaux 6.1 à 6.3.

6.6.2.1 Volume de l'espace de travail (Critère A)

Le premier critère est évidement le volume de l'espace de travail car celui-ci doit être le plus grand possible afin de maximiser les possibilités du manipulateur. La méthode pour obtenir le volume de l'espace de travail consiste à calculer le rapport entre le nombre de solutions sans collisions atteignables par rapport au nombre de solutions maximum possible dans la portion discrétisée de l'espace d'orientation. Toutefois, comme un grand espace de travail ne garantit par nécessairement que tout l'espace est utilisable aisément, il est nécessaire d'ajouter d'autres critères.

6.6.2.2 Facteur de forme de l'espace de travail (Critère B)

Le facteur de forme est calculé à partir de deux caractéristiques du l'espace de travail :

- D'une part, la valeur absolue de ψ_2 maximale est déterminée pour laquelle l'effecteur est capable de pivoter autour l'axe des Z, pour $|\psi_2|$ constant en faisant varier ψ_1 et ψ_3 si nécessaire. De manière pratique, cette valeur détermine

la dimension d'une calotte sphérique de l'espace projeté pour laquelle toutes les positions sont atteignables. Elle est intégrée au critère d'optimisation sous forme du rapport $\frac{|\psi_{2max}|}{\pi/2}$.

- D'autre part, un volume pondéré de l'espace de travail est calculé. En effet, les orientations de l'effecteur pour lesquelles la valeur absolue de ψ_2 est faible sont celles qui ont la plus grande probabilité d'être utilisées lors de l'usinage. Ainsi, cette caractéristique s'obtient en pondérant chaque solution du modèle géométrique inverse durant le calcul de l'espace de travail à l'aide de la fonction de poids suivante :

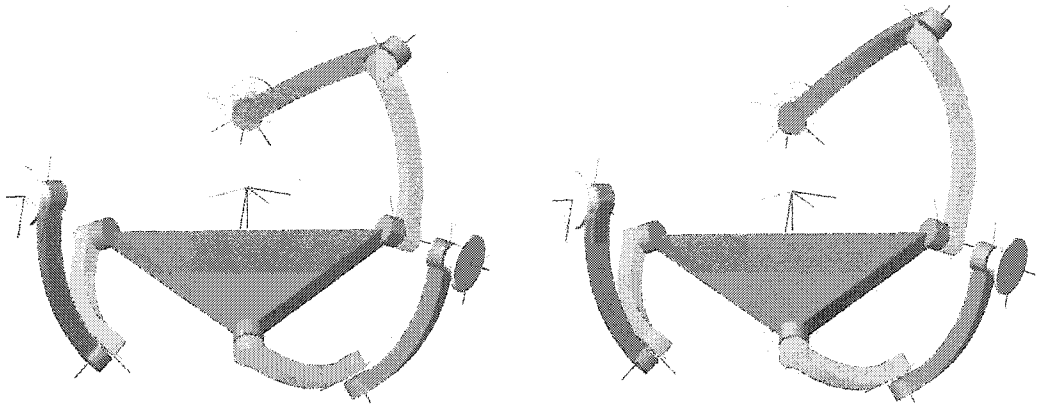
$$f_{poids}(\psi_2) = 0.35 \times \cos \psi_2 + 0.65 \tag{6.2}$$

où ψ_2 est la valeur de l'angle pour la solution concernée. Les constantes ont été choisies de manière à ce que la distribution sinusoïdale prenne la valeur 1 lorsque $\psi_2 = 0$ et la valeur 0.3 lorsque $|\psi_2| = 90°$.

La valeur du facteur de forme est une moyenne des deux caractéristiques.

6.6.2.3 Volume de l'espace de travail tenant compte de la position des jambes (Critère C)

Généralement, le manipulateur ne pourra pas être utilisé de manière optimale à l'aide d'un seul mode d'opération. En effet, l'existence des obstructions entre les membrures ne permet pas de profiter pleinement des possibilités du manipulateur. Par ailleurs, en l'absence d'obstructions, la zone de travail de la machine à commande numérique peut être grandement réduite à cause de la présence d'une jambe (Figure 6.4) du mauvais côté de l'effecteur. Idéalement, il serait pratique que toutes les jambes soient disposées en dessous de l'effecteur (Figure 6.5) mais



(a) une jambe de 100° (b) une jambe de 130°

FIGURE 6.4 – Encombrement d'une jambe en fonction de sa longueur.

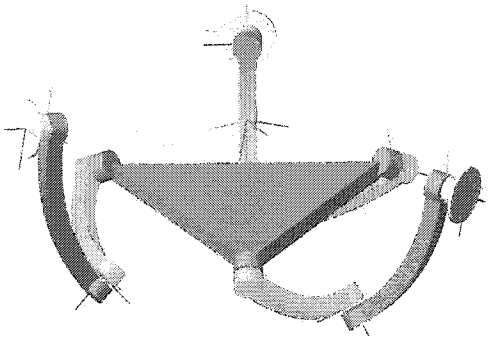


FIGURE 6.5 – Volume de travail complètement libre.

ceci n'est pas toujours possible en pratique pour les orientations où $|\psi_2|$ est grande. Une méthode simple, même si elle est incomplète, pour tenir compte de l'encombrement d'une jambe est de vérifier la position du joint intermédiaire par rapport à l'effecteur.

La figure 6.6 représente une sphère de diamètre unitaire coupée par un plan représentant l'effecteur passant par les sommets des trois vecteur unitaires $\mathbf{w}_3^k$. Le vecteur normal $\mathbf{n}$ au plan est calculé. Dans le chapitre sur la cinématique, l'axe des Z du repère rattaché à l'effecteur est orienté par le vecteur $\mathbf{w}_3$ qui est le vecteur-somme normé des trois $\mathbf{w}_3^k$. Le point G qui correspondrait à la position du référentiel de

la table d'usinage est construit de manière à ce que la droite (OG) ait comme vecteur directeur $\mathbf{w}_3$ et à ce que G appartienne au plan représentant l'effecteur. Plus l'angle φ_k entre $\mathbf{n}$ et $(\overrightarrow{GO} + \mathbf{w}_2^k)$ est petit, plus les probabilités sont grandes que la jambe k obstrue le volume de travail. Par contre, si cet angle est plus grand que 90°, la jambe est disposée en dessous de l'effecteur, à moins que le joint proximal de cette jambe ne soit positionné au-dessus de l'effecteur (mais cette situation est inévitable!).

Le critère de performance tenant compte de la position des jambes consiste à pondérer les solutions du modèle géométrique inverse obtenues lors du calcul de l'espace de travail à l'aide de la fonction de poids suivante :

$$f_{poids} = \frac{f_{poids}(\varphi_A) + f_{poids}(\varphi_B) + f_{poids}(\varphi_C)}{3} \tag{6.3}$$

avec, pour chaque jambe,

$$\begin{cases} f_{poids}(\varphi_k) = \frac{\frac{\pi}{2} - \varphi_k}{\frac{\pi}{2}} & \text{si} \quad 0 \leq \varphi_k \leq \frac{\pi}{2} \\ f_{poids}(\varphi_k) = 1 & \text{si} \quad \frac{\pi}{2} < \varphi_k \leq \pi \end{cases} \tag{6.4}$$

6.6.2.4 Degré de symétrie du manipulateur (Critère D)

Du point de vue de la fabrication de la machine, il serait plus pratique d'avoir un manipulateur symétrique. Le critère de symétrie du manipulateur consiste à calculer une moyenne du degré de symétrie du bâti, des jambes et de l'effecteur. Le degré de symétrie du bâti se calcule de la manière suivante :

$$degré\ de\ symétrie_{bâti} = \left(\frac{\min(\alpha_{AB}, \alpha_{AC}, \alpha_{BC})}{\max(\alpha_{AB}, \alpha_{AC}, \alpha_{BC})} \right)^3 \tag{6.5}$$

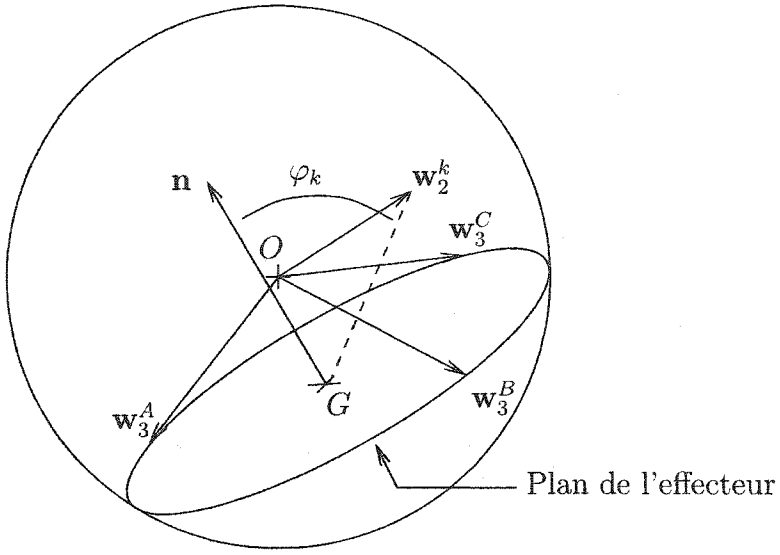


FIGURE 6.6 – Position du joint intermédiaire par rapport à l'effecteur.

Le degré de symétrie de l'effecteur se calcule de la même manière :

$$degré\ de\ symétrie_{\,effecteur} = \left(\frac{\min(\delta_{AB},\delta_{AC},\delta_{BC})}{\max(\delta_{AB},\delta_{AC},\delta_{BC})} \right)^3 \tag{6.6}$$

Enfin, celui des jambes s'obtient à l'aide de l'expression suivante :

$$degré\ de\ symétrie_{\,jambes} = \left(\frac{\min(ratio_A,ratio_B,ratio_C)}{\max(ratio_A,ratio_B,ratio_C)} \right)^3 \tag{6.7}$$

avec

$$ratio_{\,k} = \frac{\beta_k}{\beta_k + \gamma_k} \quad k = A,B,C \tag{6.8}$$

6.6.2.5 Longueur des jambes (Critère E)

Afin de limiter l'inertie et la flexion des jambes, il est avantageux d'avoir des jambes courtes. La valeur du critère de dimension des jambes grandit à mesure la longueur

des jambes diminue selon l'expression suivante :

$$critère\ de\ longueur\ des\ jambes = \frac{\frac{\pi-(\beta_A+\gamma_A)}{\pi} + \frac{\pi-(\beta_B+\gamma_B)}{\pi} + \frac{\pi-(\beta_C+\gamma_C)}{\pi}}{3} \quad (6.9)$$

6.6.2.6 Dimension du bâti (Critère F)

Plus les joints proximaux sont rapprochés, moins ils ont tendance à se retrouver du côté utile de l'effecteur et plus la valeur du critère de dimension du bâti est grand :

$$critère\ de\ dimension\ du\ bâti = \frac{2\pi - (\alpha_{AB} + \alpha_{AC} + \alpha_{BC})}{2\pi} \quad (6.10)$$

6.6.2.7 Dimension de l'effecteur (Critère G)

Un effecteur de faibles dimensions où les joints seraient rapprochés permettrait aussi de réduire l'inertie. Plus les joints distaux sont rapprochés, plus la valeur du critère de dimension de l'effecteur est grand :

$$critère\ de\ dimension\ de\ l'effecteur = \frac{2\pi - (\delta_{AB} + \delta_{AC} + \delta_{BC})}{2\pi} \quad (6.11)$$

6.6.3 Implantation des critères dans l'algorithme génétique

Le but de l'optimisation multicritères est d'obtenir des manipulateurs qui sont bons en moyenne pour l'ensemble des critères. Une combinaison linéaire des critères est donc ajoutée. Elle sert aussi à classer la population avant l'élimination des individus les moins performants. La pondération de chaque critère est choisie selon leur importance. Le tableau 6.1 présente les valeurs choisies pour le calcul présenté.

Lors du croisement, chacun des critères ainsi que la combinaison linéaire sont uti-

TABLEAU 6.1 – Pondération des différents critères dans la combinaison linéaire

	A	B	C	D	E	F	G
Poids	20%	20%	20%	15%	15%	5%	5%

A=Taille de l'espace de travail
B=Facteur de forme de l'espace de travail
C=Volume de l'espace de travail tenant compte de la position des jambes
D=Degré de symétrie du manipulateur
E=Longueur des jambes
F=Dimension du bâti
G=Dimension de l'effecteur

lisés à tour de rôle. Dans le calcul réalisé, la population comprenant 125 individus, il a été choisi de générer 25 enfants à partir d'individus performants selon la combinaison linéaire et de répartir les 100 autres enfants générés à travers les sept critères individuels en utilisant la pondération de chaque critère dans la combinaison linéaire. Ainsi, pour chaque critère, la population est classée puis des individus sont sélectionnés pour le croisement, au hasard, mais en tenant compte de leur performance relativement au critère concerné.

6.6.4 Résultats

Au bout de 208 générations et quelques jours de calcul, deux individus performants et assez semblables ont été trouvés. Ceux-ci sont décrits dans le tableau 6.2. La performance de chacun selon les différents critères et la combinaison linéaire est présentée dans le tableau 6.3.

La même remarque que pour le premier calcul d'optimisation s'applique aussi pour ce calcul ci. En effet, la précision est sûrement très inférieure au millième. Les deux individus sont donc quasiment identiques. Quelques différences sont toutefois

TABLEAU 6.2 – Paramètres géométriques des deux individus performants

	α_{AB}	α_{AC}	α_{BC}	α_{AB}	α_{AC}	α_{BC}
Individu 1	118.968°	118.881°	118.866°	120.002°	120.003°	119.995°
Individu 2	117.543°	119.401°	120.000°	118.685°	119.037°	119.998°
	β_A	γ_A	β_B	γ_B	β_C	γ_C
Individu 1	77.425°	77.549°	77.469°	77.676°	77.457°	77.698°
Individu 2	80.688°	80.342°	83.858°	86.601°	80.361°	82.586°

TABLEAU 6.3 – Performance des individus selon chaque critère

	Combi	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Individu 1	60.59%	73.41%	86.90%	57.21%	99.79%	13.84%	0.91%	0%
Individu 2	60.14%	76.24%	89.99%	59.21%	91.64%	8.44%	0.85%	0%

A=Taille de l'espace de travail
B=Facteur de forme de l'espace de travail
C=Volume de l'espace de travail tenant compte de la position des jambes
D=Degré de symétrie du manipulateur
E=Longueur des jambes
F=Dimension du bâti
G=Dimension de l'effecteur

constatées :

- L'individu 2 est moins symétrique que l'individu 1. Il aurait peut-être fallu beaucoup de générations supplémentaires pour converger encore plus vers un manipulateur symétrique. Quasiment tous les individus restants ressemblaient à l'individu 1, ce qui prouve que l'algorithme avait déjà convergé suffisamment vers une solution.
- Les deux individus ont un effecteur dont les trois angles δ_{kl} sont presque égaux à 120°. Ceci est à mettre en relation avec l'analyse qualitative de l'influence de la forme de l'effecteur sur la forme de l'espace de travail projeté. Celui-ci se creuse à mesure que la taille de l'effecteur diminue, ce qui a tendance à faire diminuer la valeur du facteur de forme de l'espace de travail dans

l'optimisation.

- Les trois angles α_{kl} du bâti sont légèrement inférieurs à 120°. Il peut s'agir d'imprécisions numériques mais à 1° de différence, l'impact sur la performance des manipulateurs est mineur.
- Les dimensions des membrures proximales et distales des deux manipulateurs ont tendance à être très proches. Avec quelques générations supplémentaires, l'écart aurait sûrement tendance à diminuer.
- Les membrures ont en moyenne une longueur $(\beta_k + \gamma_k)$ respective de 77.5° et de 82.4°pour l'individu 1 et 2 respectivement. Parmi tous les paramètres géométriques, l'écart dans les longueurs des jambes entre les deux manipulateurs est le plus marqué. Vraisemblablement que le manipulateur optimal doit se situer proche des individus présentés.

Ainsi, selon les critères de performance choisis et selon la pondération qui leur a été associée, le manipulateur optimal doit avoir des paramètres géométriques proches de ceux ci-dessous :

- $\alpha_{kl} = 119°$;
- $\beta_k = \gamma_k = 80°$;
- $\delta_{kl} = 120°$;

Bien que les critères de performance aient été différents dans les deux calculs réalisés, le résultat est quasi-identique. Les membrures sont légèrement plus courtes de 4° pour le second calcul.

6.7 Optimisation multicritères pour l'application envisagée — deuxième essai

Dans ce deuxième essai, le domaine de recherche est réduit aux manipulateurs symétriques et conformes aux conditions suivantes énoncées dans le cahier des charges de la machine-outil à commande numérique :

- Diamètre de la table : 400 mm avec rainures de fixation;
- Centre de rotation localisé à 200 mm au-dessus de la table;

Après calcul, ceci signifie que l'effecteur a une géométrie fixée : les angles δ_{kl} sont tous égaux à environ 75°.

Comme tous les manipulateurs sont symétriques et que le bâti a une géométrie fixée, seulement trois paramètres restent à optimiser : α, β et γ correspondant respectivement à la forme du bâti ($\alpha_{kl} = \alpha$), à la taille des membrures proximales ($\beta_k = \beta$) et distales ($\gamma_k = \gamma$). Deux critères d'optimisation sont aussi éliminés par rapport au calcul précédent : le degré de symétrie du manipulateur et le critère de forme de l'effecteur. La pondération des critères restants dans la combinaison linéaire est présentée au tableau 6.4.

L'opérateur de croisement ne change pas. Par contre, l'opérateur de mutation est grandement simplifié. Il consiste dorénavant à réaliser aléatoirement une des opérations suivantes sur les individus devant subir une mutation :

- Changer les dimensions du bâti;
- Déplacer le joint intermédiaire tout en conservant la même longueur totale pour les jambes;
- Changer la longueur des jambes tout en conservant la même position relative

du joint intermédiaire.

TABLEAU 6.4 – Nouvelle pondération des critères dans la combinaison linéaire

	A	B	C	E	F
Poids	20%	25%	25%	15%	15%

A=Taille de l'espace de travail
B=Facteur de forme de l'espace de travail
C=Volume de l'espace de travail tenant compte de la position des jambes
E=Longueur des jambes
F=Dimension du bâti

Au bout de 75 générations sur une population de 150 individus, le calcul a déjà convergé vers un manipulateur dont les paramètres sont :

- $\alpha = 110.080°$;
- $\beta = 70.193°$;
- $\gamma = 75.560°$;

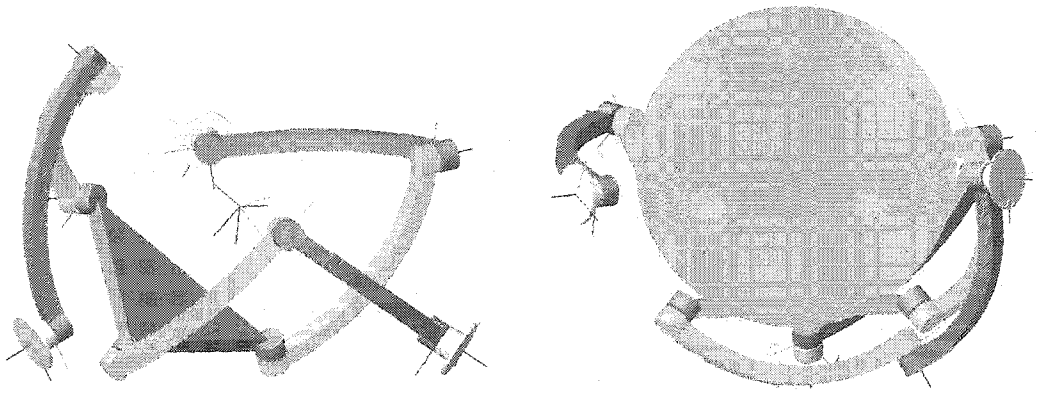
La forme du manipulateur obtenu est bien différente des deux calculs précédents. En effet, le bâti et les jambes sont bien plus petites. De plus, la longueur β_k des membrures proximales est bien différente de celle des membrures distales (γ_k), contrairement aux résultats précédents.

La performance de ce manipulateur selon chacun des critères est présentée dans le tableau 6.5. Bien que le manipulateur soit très différent de ceux obtenus précédemment, sa performance n'est pas de beaucoup inférieure. La figure 6.7 présente la forme du manipulateur ainsi que son espace de travail projeté.

TABLEAU 6.5 – Performance de l'individu optimal

Combi	A	B	C	E	F
53.065%	69.424%	86.442%	53.904%	19.026%	8.267%

A=Taille de l'espace de travail
B=Facteur de forme de l'espace de travail
C=Volume de l'espace de travail tenant compte de la position des jambes
E=Longueur des jambes
F=Dimension du bâti



(a) forme (b) espace de travail projeté

FIGURE 6.7 – Le manipulateur optimal

CONCLUSION

7.1 Travail réalisé

Le travail réalisé comporte plusieurs volets : un paramétrage minimal permettant de décrire toutes les géométries possibles des manipulateurs sphériques 3-RRR, la réalisation d'une librairie générique pour les 2^k-arbres, la mise en place d'une technique de détection de collision simple et rapide et l'intégration de l'ensemble dans un calcul d'optimisation.

Le paramétrage du manipulateur a été obtenu après plusieurs essais. Douze paramètres indépendants ont été obtenus pour décrire l'ensemble des manipulateurs sphériques de topologie 3-RRR. Un programme de simulation a permis d'étudier l'impact des différents paramètres sur la forme de l'espace de travail du manipulateur. Par ailleurs, l'optimisation ayant montré que les manipulateurs symétriques sont plus efficaces que ceux qui sont dissymétriques, le paramétrage présenté dans ce mémoire peut dorénavant être simplifié afin de ne tenir compte que des manipulateurs symétriques.

La librairie générique pour les 2^k-arbres a permis d'adapter rapidement les programmes de simulation et d'optimisation au cours de l'évolution du projet. Les objectifs fixés lors de la création de la librairie ont donc été atteints. Elle trouvera par ailleurs facilement sa place dans d'autres projets où des espaces de travail doivent être calculés de manière numérique.

La contribution dans le domaine de la détection de collision porte non seulement sur l'utilisation en robotique de techniques couramment utilisées dans la réalité virtuelle mais aussi sur la création d'une représentation filaire bien adaptée aux

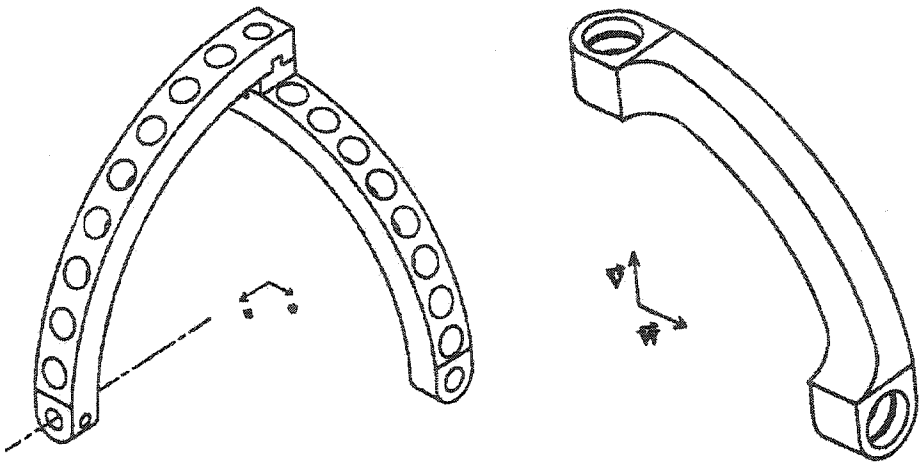
robots. À la suite de calculs comparatifs d'espaces de travail, il a été montré que la détection de collision est indispensable pour obtenir une précision convenable. Par ailleurs, la librairie créée peut s'intégrer aisément à d'autres projets grâce à sa modularité.

Les résultats obtenus par optimisation des paramètres géométriques du manipulateur sont un pas vers la réalisation de la machine à commande numérique hybride présentée en introduction. Bien que l'optimisation du manipulateur puisse être grandement améliorée, les manipulateurs optimaux générés pour l'application envisagée pourraient servir de point de départ pour la fabrication d'un prototype.

7.2 Limites du projet et perspectives

L'optimisation du manipulateur n'a porté que sur les paramètres géométriques présents dans la formulation du modèle géométrique inverse. Toutefois, les formes des membrures peuvent être modifiées sans que ces paramètres soient changés. Cette caractéristique intéressante a été utilisée dans le développement de l'œil agile afin de réduire les collisions entre les membrures proximales dans les zones importantes de l'espace de travail. Les membrures proximales sont donc d'une forme plus complexe que les membrures distales (fig. 7.1). Cette technique pourrait être mise à profit dans l'application visée par ce projet de recherche mais rendrait la tâche d'optimisation plus complexe. En effet, il faudrait rajouter un paramétrage de la forme des membrures dans l'optimisation, par exemple, à l'aide de polynômes.

Une autre modification possible pour limiter les collisions entre les membrures peut porter sur la topologie du manipulateur. Dans le développement de la poignée tridimensionnelle SHaDe, une des jambes a été remplacé par un mécanisme ayant un comportement équivalent (fig. 7.2). Ceci a eu pour effet d'éliminer les risques de



(a) membrure proximale (b) Membrure distale

FIGURE 7.1 – Formes des membrures de l'œil agile (GOSSELIN et HAMEL, 1994)

collisions liés à cette jambe du manipulateur. Modifier la topologie sans changer les degrés de libertés du manipulateur n'est pas une tâche triviale et la réaliser manuellement peut devenir fastidieuse. L'usage d'algorithmes informatiques permettant la synthèse topologique et géométrique simultanée serait idéal.

L'optimisation ayant porté exclusivement sur des critères géométriques, les contraintes d'ordre dynamique du cahier des charges présenté en introduction n'ont pas été prises en compte. Les intégrer dans les calculs modifierait certainement les résultats mais demande d'aller au-delà d'une modélisation purement cinématique du manipulateur. Par ailleurs, une analyse des trajectoires d'outil les plus courantes que la machine devra réaliser permettrait de raffiner les critères de forme de l'espace de travail.

Le passage d'un mode opératoire à un autre, qui peut être indispensable selon la trajectoire à réaliser, n'a pas été traité dans ce mémoire. Par ailleurs, ce passage se fait nécessairement en traversant une zone de singularité sérielle du manipulateur,

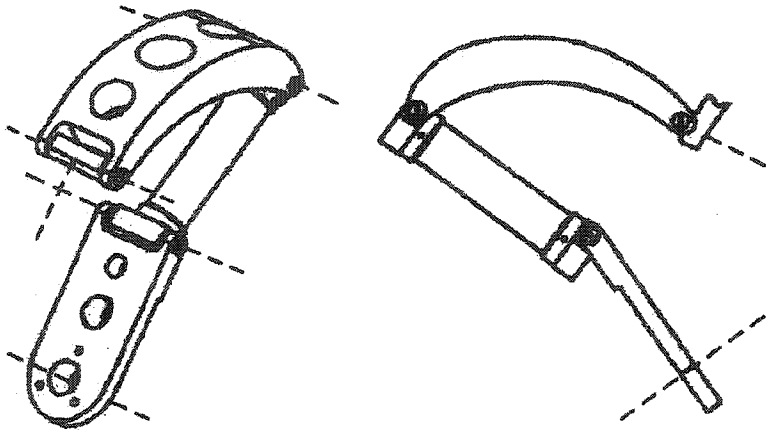


FIGURE 7.2 – Modification de la topologie d'une jambe de SHaDe (BIRGLEN et coll., 2002)

ce qui pourrait poser des problèmes supplémentaires pour le contrôle du manipulateur. Un post-processeur pour un logiciel de génération de trajectoires d'usinage serait donc un outil indispensable pour vérifier le comportement du manipulateur dans des conditions réelles d'utilisation.

Finalement, le travail de recherche présenté dans ce mémoire a été partiellement publié à deux conférences (BRUNET et BARON, 2003a; BRUNET et BARON, 2003b).

RÉFÉRENCES

ANGELES, Jorge. 1997. *Fundamentals of Robotic Mechanical Systems – theory, methods and algorithms.* Mechanical Engineering Series. Springer Verlag.

BARON, Luc. 1998. *Genetic Algorithm for Line Extraction.* Montréal : Éditions de l'École Polytechnique. EPM/RT-98/06. 20p.

BARON, Luc. 2001. « Workspace-Based Design of Parallel Manipulators of Star Topology with a Genetic Algorithm », *Proceedings of ASME Design Engineering Technical Conferences*, DETC/DAC-21026, Pittsburgh, USA.

BERNIER, Ghislain. 2003. *Synthèse de manipulateurs parallèles isotropes de topologie Star par algorithmes génétiques.* Mémoire de maîtrise en génie mécanique, École Polytechnique de Montréal.

BIRGLEN, Lionel, GOSSELIN, Clément M., POULIOT, Nicolas, MONSARRAT, Bruno, LALIBERTÉ, Thierry. 2002. « SHaDe, a new 3-dof haptic device ». *IEEE Transactions on Robotics and Automation.* 18:2. pp. 166-175.

BRUNET, Stéphane, BARON, Luc. 2003. « Calculs d'espaces de travail à l'aide d'algorithmes de détection de collision : comparaison de deux méthodes ». *Proceedings of 2003 CCToMM Symposium on Mechanisms, Machines and Mechatronics.* Commission Canadienne pour la Théorie des Machines et des Mécanismes.

BRUNET, Stéphane, BARON, Luc. 2003. « Workspace Computation of Parallel Manipulators Using 2^k-Trees with Collision Detection ». *Proceedings of the 11th World Congress in Mechanisms and Machine Science.* T. HUANG Ed. China Machinery Press.

CARRETERO, J.A., NAHON, M.A., MA, O. 2002. « Using Genetic Algorithms with Niche Formation to Solve the Minimum Distance Problem amongst Concave Objects ». *Proceedings of 28th ASME Design Automation Conference.*

CHABLAT, Damien. 1998. *Domaines d'unicité et parcourabilité pour les manipulateurs pleinement parallèles.* Thèse de doctorat en génie mécanique, École Centrale de Nantes.

CHAZELLE, Bernard, DOBKIN, David P., SHOURABOURA, Nadia, TAL, Ayellet. 1995. « Strategies for Polyhedral Surface Decomposition: An Experimental Study ». *Symposium on Computational Geometry.* pp. 297-305.

CRAIG, J.J. 1989. *Introduction to Robotics Mechanisms and Control.* Addison-Wesley.

CRAVER, W. M.. 1989. *Structural Analysis and Design of a Three-Degree-of-Freedom Robotic Shoulder Module.* Mémoire de maîtrise, University of Texas at Austin.

DANIALI, Hamid Reza Mohammadi. 1995. *Contribution to the kinematic synthesis of parallel manipulators.* Thèse de doctorat en génie mécanique, Université McGill.

DENAVIT, J., HARTENBERG, R.S. 1955. « A kinematic notation for lower-pair mechanisms based on matrices ». *ASME Journal of Applied Mechanics.* Volume 77. pp. 215-221.

DOBRJANSKYJ, L., FREUDENSTEIN, F. 1967. « Some Application of Graph Theory to the Structural Analysis of Mechanisms », *ASME Journal of Engineering for Industry*, series B. Volume 89. pp. 153-158.

EHMANN S.A., LIN M.C. 2001. « Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition ». *Eurographics.* 20:3.

FATTAH, Abbas. 1995. *Dynamics of Robotic Manipulators with Flexible Links and Kinematic Loops.* Thèse de doctorat en génie mécanique, Université McGill.

GOLDBERG, David E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

GOSSELIN, Clément. 1988. *Kinematic analysis, optimization and programming of parallel robotic manipulators.* Thèse de doctorat en génie mécanique, Université McGill.

GOSSELIN, Clément M., ANGELES, Jorge. 1989. « The Optimum Kinematic Design of a Spherical Three-Degree-of-Freedom Parallel Manipulator ». *Journal of Mechanical Design.* volume 111.

GOSSELIN, C.M., LAVOIE, E. 1993. « On the development of Spherical 3-DOF Parallel Manipulators », *Int. Journal of Robotic Research*, 12:4. pp. 394-402.

GOSSELIN, C.M., HAMEL, J.-F. 1994. « The agile eye: a high-performance three-degree-of-freedom camera-orienting device », *Proceedings of the IEEE International Conference on Robotics and Automation.* San Diego. pp. 781-786.

GOSSELIN, Clément M., SEFRIOUI, Jaouad, RICHARD, M. J. 1994. « On the Direct Kinematics of Spherical Three-Degree-of-Freedom Parallel Manipulator of General Architecture ». *Journal of Mechanical Design.* volume 116.

GOSSELIN, Clément M., SEFRIOUI, Jaouad, RICHARD, M. J. 1994. « On the Direct Kinematics of Spherical Three-Degree-of-Freedom Parallel Manipulator with a Coplanar Platform ». *Journal of Mechanical Design.* volume 116.

GOSSELIN, C.M., ST-PIERRE, E., GAGNÉ, M. 1996. « On the development of the agile eye: mechanical design, control issues and experimentation », *IEEE Robotics and Automation Society Magazine.* 3:4, pp. 29-37.

GOSSELIN, Clément M. 1999. « Static Balancing of Spherical 3-DoF Parallel Mechanisms and Manipulators ». *Journal of Robotics Research.* 18:8.

GOTTSCHALK, S., LIN, M.C., MANOCHA, D. 1996. « OBB-Tree: A Hierarchical Structure for Rapid Interference Detection ». *Proceedings of ACM SIGGRAPH.*

GOUGH, V.E., WHITEHALL, S.G.. 1962. « Universal Tyre Test Machine ». *Proceedings of 9th International Technical Congress*, F.I.S.I.T.A.. Londres, Grande-Bretagne. Volume 177.

HERRERA, Francisco, LOZANO, Manuel, VERDEGAY, Jose L. 1998. « Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis ». *Artificial Intelligence Review.* 12:4. pp. 265-319.

HERVE, J.M., SPARACINO, F. 1992. « Star : a new Concept in Robotics ». *3rd International Conference 3K-ARK*, Ferrara, Italie. pp. 176-183.

JOSUTTIS, Nicolai M. 1999. *The C++ Standard Library : A Tutorial and Reference.* Addison-Wesley. 799 p.

LIN, M.C., CANNY, J.F. 1991. « A fast algorithm for incremental distance computation ». *Proceedings of IEEE Int. Conf. on Robotics and Automation.*

LIN, M.C., GOTTSCHALK, S. 1998. « Collision Detection between Geometric Models: A Survey ». *Proceedings of IMA Conference on Mathematics of Surfaces.*

MAJOU, F., WENGER, P., CHABLAT, D. 2002. « A Novel Method for the Design of 2-DOF Parallel Mechanisms for Machining Applications », *Advances in Robot Kinematics.* J. Lenarčič and F. Thomas (Eds.), Kluwer Academic Publishers.

MEAGHER, D. 1982. « Geometric modelling using octree encoding ». *Computer Graphics and Image Processing.* Volume 19. pp. 129-147.

MORTON, G.M. 1966. « A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing ». IBM Ltd., Ottawa.

SABSABY, Mahmoud. 2002. *Synthèse d'un manipulateur parallèle sphérique à trois degrés de liberté.* Mémoire de maîtrise en génie mécanique, École Polytechnique de Montréal.

SEFRIOUI, Jaouad. 1992. *Problème géométrique direct et lieux de singularité des manipulateurs parallèles.* Thèse de doctorat en génie mécanique, Université Laval.

SPARACINO, F., HERVE, J.M. 1993. « Synthesis of Parallel Manipulators Using Lie Groups ». *IEEE-Tsukuba International Workshop on Advanced Robotics*, Japon.

WIITALA, Jared M., STANIŠIC, Michael M. 2000. « Design of an overconstrained and dextrous spherical wrist ». *Journal of Mechanical Design.* volume 122.

STEWART, D. 1965. « A platform with Six Degree of Freedom ». *Proceedings of Institution of Mechanical Engineers.* Volume 180. pp. 371-386.

TREMBLAY, Alain. 1999. *Synthèse géométrique de manipulateurs parallèles de topologie Star par algorithme génétique.* Mémoire de maîtrise en génie mécanique, École Polytechnique de Montréal.

WANG, Xiaoyu, BARON, Luc, CLOUTIER, Guy. 2003. « Design manifold of translational parallel manipulators». *Proceedings of 2003 CCToMM Symposium on Mechanisms, Machines and Mechatronics.* Commission Canadienne pour la Théorie des Machines et des Mécanismes.

WANG, Xiaoyu, BARON, Luc, CLOUTIER, Guy. 2003. « The Design of Parallel Manipulators of Delta Topology under Isotropic Constraint». *Proceedings of the 11th World Congress in Mechanisms and Machine Science.* T. HUANG Ed. China Machinery Press.

WENGER, P., CHABLAT, D. 2000. « Kinematic Analysis of a New Parallel Machine Tool : the Orthoglide », *Advances in Robot Kinematics.* J. Lenarčič and M.M. Stanišić (Eds.), Kluwer Academic Publishers, pp. 305-314 .

ANNEXE I

TECHNIQUE DE RÉSOLUTION DES ÉQUATIONS EN $\cos\theta$ ET $\sin\theta$

La résolution du problème de cinématique inverse nécessite d'utiliser une technique particulière pour isoler facilement les inconnues θ_i dans leur équation respective. La technique est basée sur les relations trigonométriques suivantes :

$$\tan\frac{\theta}{2} = \pm\sqrt{\frac{1-\cos\theta}{1+\cos\theta}} \tag{I.1}$$

$$\tan\frac{\theta}{2} = \frac{\sin\theta}{1+\cos\theta} \tag{I.2}$$

En posant au carré l'équation I.1, l'expression obtenue[1] est tout à fait comparable avec celle de l'équation I.2 :

$$\tan^2\frac{\theta}{2} = \frac{1-\cos\theta}{1+\cos\theta} \tag{I.3}$$

Il est alors possible de transformer l'équation en $\cos\theta$ et $\sin\theta$ en une équation du second degré de la forme :

$$AT^2 + BT + C = 0 \tag{I.4}$$

Avec :

$$T = \tan\frac{\theta}{2} \tag{I.5}$$

En effet, en replaçant les $\tan\frac{\theta}{2}$ par les expressions I.2 et I.3, on obtient une équation en $\cos\theta$ et $\sin\theta$ où les termes inconnus restent à être identifiés avec ceux connus

1. Tout à fait valide car le terme de droite est toujours positif.

de l'équation à résoudre :

$$\begin{aligned} & A\tfrac{1-\cos\theta}{1+\cos\theta} + B\tfrac{\sin\theta}{1+\cos\theta} + C = 0 \\ \Leftrightarrow \quad & A(1-\cos\theta) + B\sin\theta + C(1+\cos\theta) = 0 \\ \Leftrightarrow \quad & (C-A)\cos\theta + B\sin\theta + (A+C) = 0 \end{aligned} \tag{I.6}$$

Avec la condition nécessaire que $\theta \neq \pi[2\pi]$ afin que l'équivalence soit correcte[2]. La suite du problème se résout ensuite en deux temps, avec les techniques de calcul de racines des polynômes de second ordre puis avec la fonction arctan.

2. Il est alors indispensable de vérifier que ces angles ne sont pas solutions de l'équation de départ. Dans le cas contraire, il y a un risque de perdre des solutions du problème.

ANNEXE II

FICHIER MATLAB UTILISÉ POUR LE PARAMÉTRAGE ET LA CINÉMATIQUE

Cette annexe contient le fichier Matlab utilisé pour générer les équations analytiques qui ont été par la suite intégrées dans les programmes en C++. Trois fonctions retournant des matrices de rotation autour de X, Y, et Z ont été programmées :

```
function R = Rx(angle)
% Rx matrice de rotation selon x d'angle angle
R = [ 1, 0, 0 ; 0, cos(angle), -sin(angle) ; 0, sin(angle), cos(angle) ];

function R = Ry(angle)
% Ry matrice de rotation selon y d'angle angle
R = [cos(angle), 0, sin(angle) ;  0, 1, 0 ; -sin(angle), 0, cos(angle) ];

function R = Rz(angle)
% Rz matrice de rotation selon z d'angle angle
R = [cos(angle), -sin(angle), 0 ; sin(angle), cos(angle), 0 ; 0, 0, 1 ];
```

Le fichier Matlab suivant crée toutes les matrices et les expressions mathématiques des chapitres sur le paramétrage et la cinématique du manipulateur étudié :

```
% Manipulateur parallele spherique a trois degres de liberte
clear

% PARAMETRES GEOMETRIQUES
% Positionnement des actionneurs
syms alphaAC alphaAB alphaBC real;
syms alpha0A alpha0B real;
syms zetaA zetaB real;
```

```
syms zeta0 real;
% Géométrie de l'effecteur
syms deltaAC deltaAB deltaBC real;
syms delta3A delta3B real;
syms xiA xiB real;
syms xi3 real;
% Géométrie des membrures proximales
syms betaA betaB betaC real;
% Géométrie des membrures distales
syms gammaA gammaB gammaC real;

% Matrices de rotation du bati
syms D0A D0B D0C;
syms D0;

D0 = Rx(alpha0A) * Rz(sym(pi)-zeta0)
D0A = D0
D0B = D0 * Rx(alphaAB) * Rz(sym(pi)-zetaB)
D0C = D0 * Rz(zetaA - sym(pi)) * Rx(-alphaAC)

% Matrices de rotation dans l'effecteur
syms D3A D3B D3C;
syms D3;

D3 = Rx(delta3A) * Rz(sym(pi)-xi3)
D3A = D3
D3B = D3 * Rx(deltaAB) * Rz(sym(pi) - xiB)
D3C = D3 * Rz(xiA - sym(pi)) * Rx(-deltaAC)

% ANGLES (INCONNUES CINEMATIQUES)
syms theta1A theta1B theta1C real;
syms theta2A theta2B theta2C real;
syms theta3A theta3B theta3C real;

% MATRICES DE ROTATIONS ENTRE SOLIDES
syms Q01A Q01B Q01C;
syms Q12A Q12B Q12C;
syms Q23A Q23B Q23C;

Q01A = Rz(theta1A)
Q01B = Rz(theta1B)
Q01C = Rz(theta1C)

Q12A = Rx(betaA) * Rz(theta2A)
Q12B = Rx(betaB) * Rz(theta2B)
Q12C = Rx(betaC) * Rz(theta2C)

Q23A = Rx(gammaA) * Rz(theta3A)
Q23B = Rx(gammaB) * Rz(theta3B)
Q23C = Rx(gammaC) * Rz(theta3C)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MODELE GEOMETRIQUE INVERSE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Orientation de l'effecteur
syms psi1 psi2 psi3 real;
syms Q03;
Q03 = Rz(psi1) * Rx(psi2) * Rz(psi3)

syms w1A w2A w3A;
syms w1B w2B w3B;
syms w1C w2C w3C;

w1A = D0A*Q01A*[0;0;1]
w1B = D0B*Q01B*[0;0;1]
w1C = D0C*Q01C*[0;0;1]

w2A = D0A*Q01A*Q12A*[0;0;1]
w2B = D0B*Q01B*Q12B*[0;0;1]
w2C = D0C*Q01C*Q12C*[0;0;1]

syms w3A w3B w3C real;
syms xA xB xC real;
syms yA yB yC real;
syms zA zB zC real;
w3A = [xA; yA; zA];
w3B = [xB; yB; zB];
w3C = [xC; yC; zC];

syms MGIA MGIB MGIC real;
MGIA = expand(w2A' * w3A - cos(gammaA))
MGIB = expand(w2B' * w3B - cos(gammaB))
MGIC = expand(w2C' * w3C - cos(gammaC))

syms exprW3A exprW3B exprW3C real;
exprW3A = expand(Q03*D3A*[0;0;1])
exprW3B = expand(Q03*D3B*[0;0;1])
exprW3C = expand(Q03*D3C*[0;0;1])

syms expr2W3A expr2W3B expr2W3C real;
expr2W3A = expand(D0A * Q01A * Q12A * Q23A * [0;0;1])
expr2W3B = expand(D0B * Q01B * Q12B * Q23B * [0;0;1])
expr2W3C = expand(D0C * Q01C * Q12C * Q23C * [0;0;1])
```

ANNEXE III

DISPONIBILITÉ DU CODE SOURCE ET CONDITIONS D'UTILISATION

L'ensemble du code source des programmes réalisés est disponible en écrivant à l'auteur à l'adresse `stephane.brunet@polymtl.ca`. L'usage, la copie, la modification et l'intégration dans d'autres applications d'une partie ou de la totalité du code source sont autorisés du moment que le message de copyright suivant, présent au début de chaque fichier, soit conservé :

```
// ****************************************************************************
// L'usage, la distribution et la modification de cette librairie sont autorisés
// à la seule condition que ce copyright soit conservé. Veuillez notez que cette
// librairie est mise à disposition TELLE QUELLE et sans aucune garantie.
//
// This software can be freely modified, distributed and used, for any purpose,
// as long as this copyright remains. Please note that this software is provided
// "AS IS" and without warranty of any kind.
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca
// ****************************************************************************
```

Par ailleurs, afin d'alléger le mémoire, le message complet de copyright a été omis des annexes suivantes. Toutefois, le code source présenté est mis à disposition selon les mêmes conditions que celles ci-dessus.

ANNEXE IV

LIBRAIRIE C++ GÉNÉRIQUE POUR LES 2^K-ARBRES

IV.1 Code source

La librarie programmée pour les 2^k-arbres est composée de deux fichiers d'en-tête `box.h` et `path.h` à inclure dans un programme. Le fichier `path.h` est reproduit ci-dessous :

```
// -*- c++ -*-
// path.h
//
// Template class for a path container
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca

#ifndef TREE2K_PATH_H
#define TREE2K_PATH_H

#include <cstdlib>

namespace Tree2k {

template< int dim, int depth >
class Path {
public:
  Path() {
    path = 0;
  };
  Path( const Path< dim, depth > & p) {
    path = p.path;
  };
  const int readAt(const int posn) const {
    return (path >> (dim * (depth - posn))) & MASK;
  };
  void writeAt(const int posn, const int val) {
```

```
  // find the number of shifts in order to put the corresponding position
  // at bits [dim-1:0]
  int sh = dim * (depth - posn);
  // write the value at position posn
path = (path & ~(MASK << sh)) | (val << sh);
};
const Path getNeighbour(const int dir) const {
  Path p;
  p.path = path;
  if (recursiveSearch(p, dir, depth) == true)
    return p;
  else
    return *this;
};
static const int getDimension() {
  return dim;
};
static const int getDepth() {
  return depth;
};
static const Path< dim, depth > getFirstPath() {
  static const Path< dim, depth > firstPath(FIRST_PATH);
  return firstPath;
};
static const Path< dim, depth > getLastPath() {
  static const Path< dim, depth > lastPath(LAST_PATH);
  return lastPath;
};
const Path< dim, depth > & operator=( const Path< dim, depth > & p ) {
  path = p.path;
  return *this;
};
bool operator==( const Path< dim, depth > & p ) const {
  return path == p.path;
};
bool operator!=( const Path< dim, depth > & p ) const {
  return path != p.path;
};
bool operator>=( const Path< dim, depth > & p ) const {
  return path >= p.path;
};
bool operator<=( const Path< dim, depth > & p ) const {
  return path <= p.path;
};
bool operator>( const Path< dim, depth > & p ) const {
  return path > p.path;
};
bool operator<( const Path< dim, depth > & p ) const {
  return path < p.path;
};
```

```
  Path< dim, depth > operator++( ) {
    ++path;
    return *this;
  };
  Path< dim, depth > operator--( ) {
    --path;
    return *this;
  };
  Path< dim, depth > operator++( int dummy ) {
    Path< dim, depth > oldPath(*this);
    ++path;
    return oldPath;
  };
  Path< dim, depth > operator--( int dummy ) {
    Path< dim, depth > oldPath(*this);
    --path;
    return oldPath;
  };
private:
  Path(size_t p) {
    path = p;
  };
  bool recursiveSearch(Path &p, const int dir, const int posn) const ;
  static const int MASK = (1 << dim) - 1;
  static const size_t FIRST_PATH = 0;
  static const size_t LAST_PATH = (1 << (dim * depth)) - 1;
  size_t path;
};

template< int dim, int depth >
bool Path< dim, depth >::recursiveSearch(Path &p, const int dir,
                                         const int posn) const{
  int coord = p.readAt(posn);
  int absdir = abs(dir);
  int mask = (1 << (absdir - 1));
  if ( dir > 0 )
    if ((coord & mask) == 0) {
      p.writeAt( posn,coord + mask);
      return true;
    } else {
      p.writeAt(posn, coord - mask);
      if ( posn != 1)
return recursiveSearch(p, dir, posn-1);
      else
return false;
    }
  else
    if ((coord & mask) == 0) {
      p.writeAt(posn, coord + mask);
      if ( posn != 1)
```

```
return recursiveSearch(p, dir, posn-1);
      else
return false;
    } else {
      p.writeAt(posn, coord - mask);
      return true;
    };
};

} // namespace tree2k

#endif // TREE2K_PATH_H
```

Le fichier `box.h` est reproduit ci-dessous :

```
// -*- c++ -*-
// box.h
//
// Template class for a box of dimension 'dim'
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca

#ifndef TREE2K_BOX_H
#define TREE2K_BOX_H

#include <utility>
#include <vector>
#include "path.h"

using std::pair;
using std::vector;

namespace Tree2k {
  template< int dim, int depth, typename type = float >
  class Box {
  public:
    Box();
    Box( const Box< dim, depth, type > & b);
    void setBoundaries( int dir, type min, type max );
    void setMinBoundary( int dir, type min );
    void setMaxBoundary( int dir, type max );
    pair< type, type > getBoundaries( int dir ) const {
      return boundariesTbl[dir-1];
    };
```

```
  type getSize( int dir ) const {
    return sizeTbl[dir-1];
  };
  type getVolume() const {
    return volume;
  };
  vector<type> getCenterAt( const Path< dim, depth > & p ) const;
  static const int getDimension() {
    return dim;
  };
  static const int getDepth() {
    return depth;
  };
private:
  void computeVolume();
  type volume;
  type sizeTbl[dim];
  pair< type, type > boundariesTbl[dim];
};

template< int dim, int depth, typename type >
Box< dim, depth, type >::Box() {
  // Initialize tables
  int i;
  for (i=0; i < dim; i++) {
    sizeTbl[i] = 0;
    boundariesTbl[i].first = 0;
    boundariesTbl[i].second = 0;
  }
};

template< int dim, int depth, typename type >
Box< dim, depth, type >::Box( const Box< dim, depth, type > & b) {
  // Copy tables
  int i;
  for (i=0; i < dim; i++) {
    sizeTbl[i] = b.sizeTbl[i];
    boundariesTbl[i].first = b.boundariesTbl[i].first;
    boundariesTbl[i].second = b.boundariesTbl[i].second;
  }
};

template< int dim, int depth, typename type >
void Box< dim, depth, type >::setBoundaries( int dir, type min, type max ){
  sizeTbl[dir-1] = (max - min) / static_cast<type>(1 << depth);
  boundariesTbl[dir-1].first = min;
  boundariesTbl[dir-1].second = max;
  computeVolume();
};
```

```
  template< int dim, int depth, typename type >
  void Box< dim, depth, type >::setMinBoundary( int dir, type min ){
    boundariesTbl[dir-1].first = min;
    sizeTbl[dir-1] = (boundariesTbl[dir-1].second - min)
                   / static_cast<type>(1 << depth);
    computeVolume();
  };

  template< int dim, int depth, typename type >
  void Box< dim, depth, type >::setMaxBoundary( int dir, type max ){
    boundariesTbl[dir-1].second = max;
    sizeTbl[dir-1] = (max - boundariesTbl[dir-1].first)
                   / static_cast<type>(1 << depth);
    computeVolume();
  };

  template< int dim, int depth, typename type >
  vector<type> Box< dim, depth, type >::getCenterAt( const Path< dim,
                                                     depth > & p ) const {
    int i,j;
    int posn;
    vector<type> coord(dim);
    // Initialize coordinates vector
    for(j=0; j<dim; j++)
      coord[j]=0;
    // Get integer coordinates of the leaf subbox
    for (i=1; i<=depth; i++) {
      posn = p.readAt(i);
      for (j=0; j<dim; j++)
coord[j] += static_cast<type>(((posn & (1 << j)) >> j) << (depth-i));
    }
    // Compute actual coordinates of the box center
    for(j=0; j<dim; j++)
      coord[j] = sizeTbl[j] * (coord[j] + .5) + boundariesTbl[j].first;
    // Return vector
    return coord;
  };

  template< int dim, int depth, typename type >
  void Box< dim, depth, type >::computeVolume() {
    int i;
    volume = sizeTbl[0];
    for (i=1; i<dim; i++)
      volume *= sizeTbl[i];
  };

} // namespace tree2k

#endif // TREE2K_BOX_H
```

IV.2 Exemples d'utilisation

Afin d'illustrer l'usage de cette librairie, deux exemples sont maintenant présentés. Ils consistent à calculer l'espace de travail d'un manipulateur sériel plan de topologie RR et à générer un représentation de l'espace de travail sous forme de fichier EPS (*Encapsulated PostScript*). Le premier exemple fait usage du conteneur **set** pour un simple calcul d'espace de travail, sans stocker des informations particulières dans le *quadtree*. Le fichier d'en-tête du programme est reproduit ci-dessous :

```
// -*- c++ -*-
////////////////////////////////////
//       2^k tree library         //
//           EXAMPLE 1            //
//     2-dof serial manipulator   //
////////////////////////////////////
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca

#ifndef EXAMPLE1_H
#define EXAMPLE1_H 1


// Structure for joint coordinates
struct JointCoord{
  float theta1;
  float theta2;
};

// Structure for cartesian coordinate
struct CartCoord{
  float x;
  float y;
};

// Structure for the inverse kinematic solutions
struct IKSol {
  int count;
  JointCoord first;
  JointCoord second;
```

```
};

// Manipulator class
class Manipulator {
public:
  Manipulator() {
    l1 = 1.;
    l2 = 1.;
  };
  Manipulator( float newl1, float newl2 ) {
    l1 = newl1;
    l2 = newl2;
  };
  CartCoord directKinematics(const JointCoord & pos);
  IKSol inverseKinematics(const CartCoord & pos);
  inline void setGeometry(float newl1, float newl2) {
    l1 = newl1;
    l2 = newl2;
  };
  float getl1() { return l1; };
  float get21() { return l2; };
private:
  float l1, l2;
  JointCoord midjointPosToJointCoord(const CartCoord & midPos,
                                     const CartCoord & endPos);
};

#endif // EXAMPLE1_H
```

Le corps du programme est reproduit ci-dessous :

```
// -*- c++ -*-
//////////////////////////////////////
//       2^k tree library           //
//           EXAMPLE 1              //
//     2-dof serial manipulator     //
//////////////////////////////////////
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca

#include "example1.h"
#include <cmath>
#include <cstdlib>
#include <complex.h>
#include <iostream>
```

```
#include <fstream>
#include <set>
#include "box.h"
#include "path.h"

using std::cout;
using std::endl;

static const float TOLERANCE = 1e-5;
static const int DIM = 2;
static const int DEPTH = 5;

typedef Tree2k::Path< DIM, DEPTH > QuadtreePath;
typedef Tree2k::Box< DIM, DEPTH > QuadtreeBox;
typedef std::set< QuadtreePath > Quadtree;


// Manipulator class members definitions
CartCoord Manipulator::directKinematics(const JointCoord & pos){
  CartCoord cartPos;
  cartPos.x = l1 * cosf(pos.theta1) + l2 * cosf(pos.theta1 + pos.theta2);
  cartPos.y = l1 * sinf(pos.theta1) + l2 * sinf(pos.theta1 + pos.theta2);
  return cartPos;
};

IKSol Manipulator::inverseKinematics(const CartCoord & pos){
  IKSol solutions;
  CartCoord midjoint;
  float temp;
  if ((fabsf(pos.x) < TOLERANCE) && (fabsf(pos.y) < TOLERANCE)) {
    // pos.x == 0 && pos.y == 0
    if ( fabsf(l1 - l2) < TOLERANCE ) // l1 == l2
      solutions.count = -1; // Infinite number of solutions : singularity
    else // l1 != l2
      solutions.count = 0; // No solutions
  } else if (fabsf(pos.x) < TOLERANCE) { // pos.x == 0
    midjoint.y = (l1 * l1 - l2 * l2 + pos.y * pos.y) / ( 2. * pos.y );
    temp = l1 * l1 - midjoint.y * midjoint.y;
    if ( fabsf(temp) < TOLERANCE ) { // One solution
      solutions.count = 1;
      midjoint.x = 0.;
      midjoint.y = l1;
      solutions.first = midjointPosToJointCoord( midjoint, pos );
    } else if ( temp > 0 ) { // Two solutions
      solutions.count = 2;
      midjoint.x = sqrtf(temp);
      solutions.first = midjointPosToJointCoord( midjoint, pos );
      midjoint.x = -1. * midjoint.x;
      solutions.second = midjointPosToJointCoord( midjoint, pos );
    } else // No solutions
```

```
      solutions.count = 0;
  } else if (fabsf(pos.y) < TOLERANCE) { // pos.y == 0
    midjoint.x = (l1 * l1 - l2 * l2 + pos.x * pos.x) / ( 2. * pos.x);
    temp = l1 * l1 - midjoint.x * midjoint.x;
    if ( fabsf(temp) < TOLERANCE ) { // One solution
      solutions.count = 1;
      midjoint.x = l1;
      midjoint.y = 0.;
      solutions.first = midjointPosToJointCoord( midjoint, pos );
    } else if ( temp > 0 ) { // Two solutions
      solutions.count = 2;
      midjoint.y = sqrtf(temp);
      solutions.first = midjointPosToJointCoord( midjoint, pos );
      midjoint.y = -1. * midjoint.y;
      solutions.second = midjointPosToJointCoord( midjoint, pos );
    } else // No solutions
      solutions.count = 0;
  } else { // pos.x != 0 && pos.y != 0
    float a,b;
    a = (pos.x * pos.x + pos.y * pos.y + l1 * l1 - l2 * l2) / (2. * pos.x);
    b = pos.y / pos.x;
    float delta;
    delta = 4. * ( (b * b + 1) * l1 * l1 - a * a );
    if ( fabsf(delta) < TOLERANCE ) { // delta == 0, one solution
      solutions.count = 1;
      midjoint.y = ( a * b ) / ( b * b + 1 );
      midjoint.x = a - b * midjoint.y;
      solutions.first = midjointPosToJointCoord( midjoint, pos );
    } else if ( delta > 0 ) { // delta > 0, two solution
      solutions.count = 2;
      midjoint.y = ( 2. * a * b + sqrtf(delta)) / ( 2. * ( b * b + 1 ));
      midjoint.x = a - b * midjoint.y;
      solutions.first = midjointPosToJointCoord( midjoint, pos );
      midjoint.y = ( 2. * a * b - sqrtf(delta)) / ( 2. * ( b * b + 1 ));
      midjoint.x = a - b * midjoint.y;
      solutions.second = midjointPosToJointCoord( midjoint, pos );
    } else // delta < 0, no solution
      solutions.count = 0;
  };
  return solutions;
};

JointCoord Manipulator::midjointPosToJointCoord(const CartCoord & midPos,
                                                const CartCoord & endPos){
  JointCoord coord;
  coord.theta1 = atan2f( midPos.y, midPos.x );
  float dx,dy;
  dx = endPos.x - midPos.x;
  dy = endPos.y - midPos.y;
  coord.theta2 = atan2f( midPos.x * dy - midPos.y * dx ,
```

```
                        midPos.x * dx + midPos.y * dy );
  return coord;
};


int main() {
  static const float L1 = .3;
  static const float L2 = .7;
  static const float BOX_DIM = (L1 + L2) * (2. + powf(2., -1. * (DEPTH - 1)));
  // The manipulator
  Manipulator manip(L1, L2);
  // Boundaries for quadtree computation
  QuadtreeBox box;
  box.setBoundaries(1, BOX_DIM * -.5, BOX_DIM * .5 );
  box.setBoundaries(2, BOX_DIM * -.5, BOX_DIM * .5 );
  // Quadtree data structure
  Quadtree tree;
  // Populating structure
  QuadtreePath path;
  IKSol sol;
  CartCoord pos;
  vector< float > center;
  for (path = QuadtreePath::getFirstPath(); path <= QuadtreePath::getLastPath();
       path++) {
    center = box.getCenterAt(path);
    pos.x = center[0];
    pos.y = center[1];
    sol = manip.inverseKinematics(pos);
    if ((sol.count == 1) || (sol.count == 2))
      tree.insert(path);
  }
  ////////////////////////////////
  // Writes results in an EPS file
  ofstream outputEPS;
  static const float SIZE = 3 * 72; // in points
  static const int OFFSET = 3; // in points
  static const float LINEWIDTH = .002;
  outputEPS.open( "example1.eps", ios::out );
  if ( !outputEPS ) {
    std::cerr << "File could not be opened!" << endl;
    exit(1);
  }
  outputEPS << "%!PS-Adobe-2.0 EPSF-1.2" << endl
    << "%%BoundingBox: 0 0 " << SIZE + 2 * OFFSET << " "
            << SIZE + 2 * OFFSET << endl
    << "%%EndComments" << endl;
  // A new command to display a box
  outputEPS << "/box {" << endl
    << "2 copy translate" << endl
    << "newpath" << endl
```

```
  << "dx 2 div dy 2 div moveto" << endl
  << "-dx -dy rlineto" << endl
  << "dx   0 rmoveto" << endl
  << "-dx  dy rlineto" << endl
  << "stroke" << endl
  << "newpath" << endl
  << "dx 2 div dy 2 div moveto" << endl
  << "0 -dy rlineto" << endl
  << "-dx   0 rlineto" << endl
  << "0  dy rlineto" << endl
  << "closepath" << endl
  << "stroke" << endl
  << "neg exch neg exch translate" << endl
  << "} bind def" << endl;
// A new command to display an arrow
outputEPS << "/arrow {" << endl
  << "newpath" << endl
  << "0 0 moveto" << endl
  << SIZE / 6. << " 0 lineto" << endl
  << "stroke" << endl
  << "newpath" << endl
  << SIZE / 6. << " 0 moveto" << endl
  << SIZE / -40. << " " << SIZE / 80. << " rlineto" << endl
  << "0 " << SIZE / -40. << " rlineto" << endl
  << "closepath" << endl
  << "fill" << endl
  << "} bind def" << endl;
float d;
d =  box.getSize(1);
outputEPS << "/dx " << d << " def" << endl;
outputEPS << "/-dx " << -1 * d << " def" << endl;
outputEPS << "/dy " << d << " def" << endl;
outputEPS << "/-dy " << -1 * d << " def" << endl;
outputEPS << SIZE / 2. + OFFSET << " dup translate" << endl
  << LINEWIDTH * 400 << " setlinewidth" << endl
  << "arrow 90 rotate arrow -90 rotate" << endl
  << SIZE / BOX_DIM << " dup scale" << endl
  << LINEWIDTH * 4 << " setlinewidth" << endl
  << "newpath" << endl
  << d * (1 << (DEPTH - 1)) << " dup moveto" << endl
  << "0 " << d * (1 << DEPTH) << " neg rlineto" << endl
  << d * (1 << DEPTH) << " neg 0 rlineto" << endl
  << "0 " << d * (1 << DEPTH) << " rlineto" << endl
  << "closepath" << endl
  << "stroke" << endl
  << LINEWIDTH << " setlinewidth" << endl;
// Display boxes
Quadtree::iterator node;
for (node = tree.begin(); node != tree.end(); ++node) {
  center = box.getCenterAt(*node);
```

```
    outputEPS << center[0]  << " " << center[1] << " box" << endl;
  }
  // Display actual solution
  float radius;
  radius = L1 + L2;
  float theta;
  static const float INCREMENT = M_PI * 2 / 80;
  outputEPS << LINEWIDTH * 4 << " setlinewidth" << endl
    << "newpath" << endl
    << radius << " 0 moveto" << endl;
  for (theta = INCREMENT; theta < (2. * M_PI); theta += INCREMENT)
    outputEPS << radius * cosf(theta) << " " << radius * sinf(theta)
              << " lineto" << endl;
  outputEPS << "closepath stroke" << endl;
  if ( fabsf(L1 - L2) > TOLERANCE ) { // L1 != L2
    radius = fabsf( L1 - L2 );
    outputEPS << "newpath" << endl
      << radius << " 0 moveto" << endl;
    for (theta = INCREMENT; theta < (2. * M_PI); theta += INCREMENT)
      outputEPS << radius * cosf(theta) << " " << radius * sinf(theta)
                << " lineto" << endl;
    outputEPS << "closepath stroke" << endl;
  }
  outputEPS << "showpage" << endl;
}
```

Le deuxième utilise un conteneur `map` pour stocker les différentes configurations que le manipulateur peut avoir pour une position donnée (*coude haut*, *coude base* ou les deux). Des limites articulaire ont aussi été fixées. Le fichier d'en-tête du programme est reproduit ci-dessous :

```
// -*- c++ -*-
////////////////////////////////////
//       2^k tree library         //
//           EXAMPLE 2            //
//     2-dof serial manipulator   //
////////////////////////////////////
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca

#ifndef EXAMPLE2_H
#define EXAMPLE2_H 1
```

```
#include <cmath>

// Structure for joint coordinates
struct JointCoord{
  float theta1;
  float theta2;
};

// Structure for cartesian coordinate
struct CartCoord{
  float x;
  float y;
};

// Structure for the inverse kinematic solutions
struct IKSol {
  int count;
  JointCoord first;
  JointCoord second;
};

// Enumeration for manipulator configuration
enum Config { ELBOW_UP, ELBOW_DOWN, BOTH };

// Manipulator class
class Manipulator {
public:
  Manipulator() {
    l1 = 1.;
    l2 = 1.;
    minLimit.theta1 = -1. * M_PI;
    minLimit.theta2 = -1. * M_PI;
    maxLimit.theta1 = M_PI;
    maxLimit.theta2 = M_PI;
  };
  Manipulator( float newl1, float newl2 ) {
    l1 = newl1;
    l2 = newl2;
    minLimit.theta1 = -1. * M_PI;
    minLimit.theta2 = -1. * M_PI;
    maxLimit.theta1 = M_PI;
    maxLimit.theta2 = M_PI;
  };
  CartCoord directKinematics(const JointCoord & pos);
  IKSol inverseKinematics(const CartCoord & pos);
  void setGeometry(float newl1, float newl2) {
    l1 = newl1;
    l2 = newl2;
  };
```

```
  void setJointsLimits(JointCoord newMinLimit, JointCoord newMaxLimit) {
    minLimit = newMinLimit;
    maxLimit = newMaxLimit;
  };
  inline float getl1() { return l1; };
  inline float get21() { return l2; };
  inline JointCoord getMinLimit() { return minLimit; };
  inline JointCoord getMaxLimit() { return maxLimit; };
private:
  float l1, l2;
  JointCoord minLimit, maxLimit;
  JointCoord midjointPosToJointCoord(const CartCoord & midPos,
                                     const CartCoord & endPos);
  bool inJointsLimits(const JointCoord & pos);
};

#endif // EXAMPLE2_H
```

Le corps du programme est reproduit ci-dessous :

```
// -*- c++ -*-
////////////////////////////////////
//       2^k tree library         //
//           EXAMPLE 2            //
//     2-dof serial manipulator   //
////////////////////////////////////
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca

#include "example2.h"
#include <cmath>
#include <cstdlib>
#include <complex.h>
#include <iostream>
#include <fstream>
#include <map>
#include "box.h"
#include "path.h"

using std::cout;
using std::endl;

static const float TOLERANCE = 1e-5;
static const int DIM = 2;
```

```
static const int DEPTH = 6;

typedef Tree2k::Path< DIM, DEPTH > QuadtreePath;
typedef Tree2k::Box< DIM, DEPTH > QuadtreeBox;
typedef std::map< QuadtreePath, Config > Quadtree;


// Manipulator class members definitions
CartCoord Manipulator::directKinematics(const JointCoord & pos){
  CartCoord cartPos;
  cartPos.x = l1 * cosf(pos.theta1) + l2 * cosf(pos.theta1 + pos.theta2);
  cartPos.y = l1 * sinf(pos.theta1) + l2 * sinf(pos.theta1 + pos.theta2);
  return cartPos;
};

IKSol Manipulator::inverseKinematics(const CartCoord & pos){
  IKSol solutions;
  CartCoord midjoint;
  float temp;
  if ((fabsf(pos.x) < TOLERANCE) && (fabsf(pos.y) < TOLERANCE)) {
    // pos.x == 0 && pos.y == 0
    if ( fabsf(l1 - l2) < TOLERANCE ) // l1 == l2
      solutions.count = -1; // Infinite number of solutions : singularity
    else // l1 != l2
      solutions.count = 0; // No solutions
  } else if (fabsf(pos.x) < TOLERANCE) { // pos.x == 0
    midjoint.y = (l1 * l1 - l2 * l2 + pos.y * pos.y) / ( 2. * pos.y );
    temp = l1 * l1 - midjoint.y * midjoint.y;
    if ( fabsf(temp) < TOLERANCE ) { // One solution
      solutions.count = 1;
      midjoint.x = 0.;
      midjoint.y = l1;
      solutions.first = midjointPosToJointCoord( midjoint, pos );
    } else if ( temp > 0 ) { // Two solutions
      solutions.count = 2;
      midjoint.x = sqrtf(temp);
      solutions.first = midjointPosToJointCoord( midjoint, pos );
      midjoint.x = -1. * midjoint.x;
      solutions.second = midjointPosToJointCoord( midjoint, pos );
    } else // No solutions
      solutions.count = 0;
  } else if (fabsf(pos.y) < TOLERANCE) { // pos.y == 0
    midjoint.x = (l1 * l1 - l2 * l2 + pos.x * pos.x) / ( 2. * pos.x);
    temp = l1 * l1 - midjoint.x * midjoint.x;
    if ( fabsf(temp) < TOLERANCE ) { // One solution
      solutions.count = 1;
      midjoint.x = l1;
      midjoint.y = 0.;
      solutions.first = midjointPosToJointCoord( midjoint, pos );
    } else if ( temp > 0 ) { // Two solutions
```

```
    solutions.count = 2;
    midjoint.y = sqrtf(temp);
    solutions.first = midjointPosToJointCoord( midjoint, pos );
    midjoint.y = -1. * midjoint.y;
    solutions.second = midjointPosToJointCoord( midjoint, pos );
  } else // No solutions
    solutions.count = 0;
} else { // pos.x != 0 && pos.y != 0
  float a,b;
  a = (pos.x * pos.x + pos.y * pos.y + l1 * l1 - l2 * l2) / (2. * pos.x);
  b = pos.y / pos.x;
  float delta;
  delta = 4. * ( (b * b + 1) * l1 * l1 - a * a );
  if ( fabsf(delta) < TOLERANCE ) { // delta == 0, one solution
    solutions.count = 1;
    midjoint.y = ( a * b ) / ( b * b + 1 );
    midjoint.x = a - b * midjoint.y;
    solutions.first = midjointPosToJointCoord( midjoint, pos );
  } else if ( delta > 0 ) { // delta > 0, two solution
    solutions.count = 2;
    midjoint.y = ( 2. * a * b + sqrtf(delta)) / ( 2. * ( b * b + 1 ));
    midjoint.x = a - b * midjoint.y;
    solutions.first = midjointPosToJointCoord( midjoint, pos );
    midjoint.y = ( 2. * a * b - sqrtf(delta)) / ( 2. * ( b * b + 1 ));
    midjoint.x = a - b * midjoint.y;
    solutions.second = midjointPosToJointCoord( midjoint, pos );
  } else // delta < 0, no solution
    solutions.count = 0;
};
// Check if solutions are inside joints limits
switch (solutions.count) {
case 2:
  if (inJointsLimits(solutions.first) && !inJointsLimits(solutions.second))
    solutions.count--;
  else if (!inJointsLimits(solutions.first) &&
           inJointsLimits(solutions.second)) {
    solutions.count--;
    solutions.first = solutions.second;
  } else if (!inJointsLimits(solutions.first) &&
             !inJointsLimits(solutions.second))
    solutions.count = 0;
  break;
case 1:
  if (!inJointsLimits(solutions.first))
    solutions.count = 0;
  break;
default:
  // Do nothing
  break;
};
```

```
  return solutions;
};

JointCoord Manipulator::midjointPosToJointCoord(const CartCoord & midPos,
                                                const CartCoord & endPos){
  JointCoord coord;
  coord.theta1 = atan2f( midPos.y, midPos.x );
  float dx,dy;
  dx = endPos.x - midPos.x;
  dy = endPos.y - midPos.y;
  coord.theta2 = atan2f( midPos.x * dy - midPos.y * dx ,
                         midPos.x * dx + midPos.y * dy );
  return coord;
};

bool Manipulator::inJointsLimits(const JointCoord & pos) {
  return ((pos.theta1 >= minLimit.theta1) && (pos.theta1 <= maxLimit.theta1) &&
          (pos.theta2 >= minLimit.theta2) && (pos.theta2 <= maxLimit.theta2));
};

int main() {
  static const float L1 = .5;
  static const float L2 = .6;
  JointCoord minLimit, maxLimit;
  static const float BOX_DIM = (L1 + L2) * (2. + powf(2., -1. * (DEPTH - 1)));
  // The manipulator
  Manipulator manip(L1, L2);
  minLimit.theta1 = -1. * M_PI_2;
  minLimit.theta2 = -2. * M_PI / 3;
  maxLimit.theta1 = M_PI_2;
  maxLimit.theta2 = 2. * M_PI / 3;
  manip.setJointsLimits(minLimit, maxLimit);
  // Boundaries for quadtree computation
  QuadtreeBox box;
  box.setBoundaries(1, BOX_DIM * -.5, BOX_DIM * .5 );
  box.setBoundaries(2, BOX_DIM * -.5, BOX_DIM * .5 );
  // Quadtree data structure
  Quadtree tree;
  // Populating structure
  QuadtreePath path;
  IKSol sol;
  CartCoord pos;
  vector< float > center;
  for (path = QuadtreePath::getFirstPath(); path <= QuadtreePath::getLastPath();
       path++) {
    center = box.getCenterAt(path);
    pos.x = center[0];
    pos.y = center[1];
    sol = manip.inverseKinematics(pos);
    switch (sol.count) {
```

```
    case 1:
      if ( fabsf(sol.first.theta2) < TOLERANCE ) // theta2 == 0
tree[path] = BOTH; // straight arm
      else
tree[path] = ( sol.first.theta2 > 0 ? ELBOW_DOWN : ELBOW_UP);
      break;
    case 2:
      tree[path] = BOTH;
      break;
    default:
      // Do nothing
      break;
    };
  }
  ////////////////////////////////
  // Writes results in an EPS file
  ofstream outputEPS;
  static const float SIZE = 3 * 72; // in points
  static const int OFFSET = 3; // in points
  static const float LINEWIDTH = .002;
  outputEPS.open( "example2.eps", ios::out );
  if ( !outputEPS ) {
    std::cerr << "File could not be opened!" << endl;
    exit(1);
  }
  outputEPS << "%!PS-Adobe-2.0 EPSF-1.2" << endl
    << "%%BoundingBox: 0 0 " << SIZE + 2 * OFFSET << " "
          << SIZE + 2 * OFFSET << endl
    << "%%EndComments" << endl;
  // A new command to display a filled box
  outputEPS << "/box {" << endl
    << "setgray" << endl
    << "2 copy translate" << endl
    << "newpath" << endl
    << "dx 2 div dy 2 div moveto" << endl
    << "0 -dy rlineto" << endl
    << "-dx   0 rlineto" << endl
    << "0  dy rlineto" << endl
    << "closepath" << endl
    << "fill" << endl
    << "0.0 setgray" << endl
    << "newpath" << endl
    << "dx 2 div dy 2 div moveto" << endl
    << "0 -dy rlineto" << endl
    << "-dx   0 rlineto" << endl
    << "0  dy rlineto" << endl
    << "closepath" << endl
    << "stroke" << endl
    << "neg exch neg exch translate" << endl
    << "} bind def" << endl;
```

```
// A new command to display an arrow
outputEPS << "/arrow {" << endl
  << "newpath" << endl
  << "0 0 moveto" << endl
  << SIZE / 6. << " 0 lineto" << endl
  << "stroke" << endl
  << "newpath" << endl
  << SIZE / 6. << " 0 moveto" << endl
  << SIZE / -40. << " " << SIZE / 80. << " rlineto" << endl
  << "0 " << SIZE / -40. << " rlineto" << endl
  << "closepath" << endl
  << "fill" << endl
  << "} bind def" << endl;
float d;
d =  box.getSize(1);
outputEPS << "/dx " << d << " def" << endl;
outputEPS << "/-dx " << -1 * d << " def" << endl;
outputEPS << "/dy " << d << " def" << endl;
outputEPS << "/-dy " << -1 * d << " def" << endl;
outputEPS << SIZE / 2. + OFFSET << " dup translate" << endl
  << LINEWIDTH * 400 << " setlinewidth" << endl
  << "arrow 90 rotate arrow -90 rotate" << endl
  << SIZE / BOX_DIM << " dup scale" << endl
  << LINEWIDTH * 4 << " setlinewidth" << endl
  << "newpath" << endl
  << d * (1 << (DEPTH - 1)) << " dup moveto" << endl
  << "0 " << d * (1 << DEPTH) << " neg rlineto" << endl
  << d * (1 << DEPTH) << " neg 0 rlineto" << endl
  << "0 " << d * (1 << DEPTH) << " rlineto" << endl
  << "closepath" << endl
  << "stroke" << endl
  << LINEWIDTH << " setlinewidth" << endl;
// Display boxes
Quadtree::iterator node;
for (node = tree.begin(); node != tree.end(); ++node) {
  center = box.getCenterAt(node->first);
  outputEPS << center[0]  << " " << center[1] << endl;
  switch (node->second) {
  case ELBOW_DOWN:
    outputEPS << 0.5;
    break;
  case ELBOW_UP:
    outputEPS << 1;
    break;
  case BOTH:
    outputEPS << 0.75;
    break;
  default:
    // Do Nothing
    break;
```

```
    };
    outputEPS << " box" << endl;
  }
  outputEPS << "showpage" << endl;
}
```

ANNEXE V

LIBRAIRIE C++ POUR LA DÉTECTION DE COLLISION AVEC DES MODÈLES FILAIRES

V.1 Code source

La librairie est composée de plusieurs fichiers :

- `wireframe.h`,
- `wireframe.cpp`,
- `util.h`,
- `util.cpp`.

Le fichier d'en-tête `wireframe.h` doit être inclus dans le code source du programme. Il est reproduit ci-dessous :

```
// -*- c++ -*-
////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//      Collision detection library using wireframe representation            //
//      file : wireframe.h                                                    //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca

#ifndef WIREFRAME_H
#define WIREFRAME_H 1

#include "util.h"
#include <cstddef>
```

```
struct WfPoint {
  Vector3D pt;
  double radius;
};

// The binary tree data structure
struct WfNode {
  WfNode() {
    child1 = NULL;
    child2 = NULL;
  }
  ~WfNode() {
    if (child1 != NULL)
      delete child1;
    if (child2 != NULL)
      delete child2;
  }
  bool isLeafNode() const {
    if ((child1 == NULL) && (child2 == NULL))
      return true;
    else
      return false;
  }
  int begin, end;
  double radius;
  WfNode * child1;
  WfNode * child2;
};

// Function needed to build the hierarchy
// Return the root node
WfNode * buildWfHierarchy(WfPoint points [], int begin, int end);

// Function to check for collisions
// return true if there is a collision
bool checkWfCollision(WfPoint pointsA [], WfNode * hierA, WfPoint pointsB [],
                      WfNode * hierB);

// Function to compute the radius for a fictious segment
double computeRadius(WfPoint points [], int begin, int end);
#endif // WIREFRAME_H
```

Le fichier `wireframe.cpp` est reproduit ci-dessous :

```
// -*- c++ -*-
////////////////////////////////////////////////////////////////////////////////
//                                                                            //
//      Collision detection library using wireframe representation            //
//      file : wireframe.cpp                                                  //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
//
// (c) 2002-2003 Stéphane Brunet
// stephane.brunet@polymtl.ca

#include "wireframe.h"
#include <cmath>
#include <iostream>

using namespace std;
#include <cstddef>

double computeRadius(WfPoint points [], int begin, int end) {
  Vector3D vect, tempVect;
  vect = points[end].pt - points[begin].pt;
  int i;
  double radius, maxRadius;
  vect = vect / sqrt(dot(vect, vect));
  maxRadius = 0.0;
  for (i=begin+1; i <= end; i++) {
    tempVect = points[i].pt - points[begin].pt;
    tempVect = tempVect * vect;
    radius = sqrt(dot(tempVect, tempVect));
    if (points[i-1].radius > points[i].radius)
      radius += points[i-1].radius;
    else
      radius += points[i].radius;
    if (radius > maxRadius)
      maxRadius = radius;
  }
  return maxRadius;
}

WfNode * buildWfHierarchy(WfPoint points [], int begin, int end) {
  WfNode * node = new WfNode;
  if (node->child1 != NULL)
    cout << "child1 not NULL!" << endl;
  if (node->child2 != NULL)
    cout << "child2 not NULL!" << endl;
  // set root node properties
```

```
  node->begin = begin;
  node->end = end;
  if ((end-begin) == 1)
    node->radius = points[begin].radius;
  else
    node->radius = computeRadius(points, begin, end);
  // create children
  if ((end - begin) >= 2) {
    int mid = begin + (end - begin) / 2;
    node->child1 = buildWfHierarchy(points, begin, mid);
    node->child2 = buildWfHierarchy(points, mid, end);
  }
  // return node
  return node;
}

bool checkWfCollision(WfPoint pointsA [], WfNode * hierA, WfPoint pointsB [],
                      WfNode * hierB) {
  // Compute distance between nodes at the current hierarchy level
  Segment3D s0, s1;
  s0.p0 = pointsA[hierA->begin].pt;
  s0.p1 = pointsA[hierA->end].pt;
  s1.p0 = pointsB[hierB->begin].pt;
  s1.p1 = pointsB[hierB->end].pt;
  double sqDist = squaredDistance (s0, s1);
  bool check1, check2, check3, check4;
  if (sqDist > ((hierA->radius+hierB->radius) * (hierA->radius+hierB->radius)))
    return false;
  else if (hierA->isLeafNode() && hierB->isLeafNode())
    return true;
  else if (hierA->isLeafNode()) {
    check1 = checkWfCollision(pointsA, hierA, pointsB, hierB->child1);
    check2 = checkWfCollision(pointsA, hierA, pointsB, hierB->child2);
    return check1 || check2;
  } else if (hierB->isLeafNode()) {
    check1 = checkWfCollision(pointsA, hierA->child1, pointsB, hierB);
    check2 = checkWfCollision(pointsA, hierA->child2, pointsB, hierB);
    return check1 || check2;
  } else { // check for each children
    check1 = checkWfCollision(pointsA, hierA->child1, pointsB, hierB->child1);
    check2 = checkWfCollision(pointsA, hierA->child2, pointsB, hierB->child1);
    check3 = checkWfCollision(pointsA, hierA->child1, pointsB, hierB->child2);
    check4 = checkWfCollision(pointsA, hierA->child2, pointsB, hierB->child2);
    return check1 || check2 || check3 || check4;
  }
}
```

Les fichiers `util.h` et `util.cpp` contiennent un ensemble de classes et fonctions utilisables pour d'autres tâches. Les portions importantes pour la librairie de détection de collision sont reproduites ci-dessous. Pour `util.h`:

```
// -*- c++ -*-

#ifndef UTIL_H
#define UTIL_H 1

#include <cmath>

// Requested precision of floating-point mathematics
static const double PRECISION = 1e-6;

// Structures and functions for collision detection
struct Vector3D {
  double x;
  double y;
  double z;
  Vector3D(double xx = 0, double yy = 0, double zz=0);
  Vector3D operator+(const Vector3D &v) const;
  Vector3D operator-(const Vector3D &v) const;
  Vector3D operator*(const double d) const;
  Vector3D operator*(const Vector3D &v) const;
  Vector3D operator/(const double d) const;
};

const double dot(const Vector3D v0, const Vector3D v1);

struct Segment3D {
  Segment3D( const Vector3D &pt0, const Vector3D &pt1) : p0(pt0), p1(pt1) {}
  Segment3D(){}
  Vector3D p0;
  Vector3D p1;
};

const double squaredDistance (const Segment3D &s0, const Segment3D &s1);

#endif // UTIL_H
```

Et `util.cpp`:

```
// -*- c++ -*-

#include "util.h"
#include <cmath>

// Structures and functions for collision detection
Vector3D::Vector3D(double xx, double yy, double zz) {
  x = xx;
  y = yy;
  z = zz;
}
Vector3D Vector3D::operator+(const Vector3D &v) const {
  Vector3D sol;
  sol.x = x + v.x;
  sol.y = y + v.y;
  sol.z = z + v.z;
  return sol;
}
Vector3D Vector3D::operator-(const Vector3D &v) const {
  Vector3D sol;
  sol.x = x - v.x;
  sol.y = y - v.y;
  sol.z = z - v.z;
  return sol;
}
Vector3D Vector3D::operator*(const double d) const {
  Vector3D sol;
  sol.x = x * d;
  sol.y = y * d;
  sol.z = z * d;
  return sol;
}

Vector3D Vector3D::operator*(const Vector3D &v) const {
  Vector3D sol;
  sol.x = y * v.z - z * v.y;
  sol.y = z * v.x - x * v.z;
  sol.z = x * v.y - y * v.x;
  return sol;
}

Vector3D Vector3D::operator/(const double d) const {
  Vector3D sol;
  sol.x = x / d;
  sol.y = y / d;
  sol.z = z / d;
```

```
  return sol;
}

const double dot(const Vector3D v0, const Vector3D v1) {
  return v0.x * v1.x + v0.y * v1.y + v0.z * v1.z;
}

const double squaredDistance (const Segment3D &s0, const Segment3D &s1) {
  // Convert the representation in a usable form
  Vector3D d0 = s0.p1 - s0.p0;
  Vector3D d1 = s1.p1 - s1.p0;
  // Compute some values
  Vector3D u = s1.p0 - s0.p0;
  double a = dot(d0, d0);
  double b = dot(d0, d1);
  double c = dot(d1, d1);
  double d = dot(d0, u);
  double e = dot(d1, u);
  double det = a * c - b * b;
  double sNum, tNum, sDenom, tDenom;
  // Compute mininum distance between lines
  if (isNull(det)) { // The parallel case
    tNum = 0;
    tDenom = 0;
    sNum = d;
    sDenom = a;
  } else { // The non-parallel case
    sNum = c * d - b * e;
    tNum = b * d - a * e;
    sDenom = det;
    tDenom = det;
  }
  // Check if outside of the [0,1] x [0,1] domain
  if (sNum < 0) {
    sNum = 0;
    sDenom = 1.; // arbitrary choice
    tNum = -1. * e;
    tDenom = c;
  } else if (sNum > sDenom) {
    sNum = 1.;
    sDenom = 1.; // arbitrary choice
    tNum = b - e;
    tDenom = c;
  } else if (tNum < 0) {
    tNum = 0;
    tDenom = 1; // arbitrary choice
    sNum = d;
    sDenom = a;
  } else if (tNum > tDenom) {
    tNum = 1;
```

```
    tDenom = 1; // arbitrary choice
    sNum = b + d;
    sDenom = a;
  }
  if (sNum < 0)
    sNum = 0;
  else if (sNum > sDenom)
    sNum = sDenom;
  if (tNum < 0)
    tNum = 0;
  else if (tNum > tDenom)
    tNum = tDenom;
  Vector3D v = s0.p0 + d0 * (sNum / sDenom) - s1.p0 - d1 * (tNum / tDenom);
  return dot(v, v);
}
```

V.2 Exemple d'utilisation

Afin d'illustrer l'usage de la librairie, la marche à suivre est présentée ci-dessous. Le modèle filaire de chaque jambe est contenu dans un tableau :

```
WfPoint legA [NB_SEGMENTS+1], legB [NB_SEGMENTS+1];
```

Chaque sommet du modèle filaire est une structure comportant les coordonnées du sommet ainsi que le rayon du segment partant de ce point. C'est-à-dire que le premier segment de matière a un rayon égal à celui donné pour le premier sommet et ainsi de suite... Le rayon donné pour le dernier sommet n'a donc aucune influence puisqu'il ne sera pas utilisé pour aucun segment. Avant de lancer la décomposition hiérarchique, il faut créer une variable de type `WfNode *` par solide qui correspondra ensuite à la racine de la hiérarchie générée :

```
WfNode * hLegA, * hLegB;
```

Pour générer la hiérarchie, il suffit d'un appel de fonction :

```
hLegA = buildWfHierarchy(legA, 0, NB_SEGMENTS);
hLegB = buildWfHierarchy(legB, 0, NB_SEGMENTS);
```

Bien que les jambes se déplacent dans l'espace, la décomposition hiérarchique ne change pas. Ainsi, n'importe quelle translation et/ou rotation d'une jambe n'oblige pas de régénérer la hiérarchie. Après avoir réaliser la transformation voulue sur les tableaux `legA` et `legB`, la requête de détection de collision, qui retourne une valeur booléenne, s'appelle de la manière suivante :

```
checkWfCollision(legA, hLegA, legB, hLegB)
```